# VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií


# DIPLOMOVÁ PRÁCE


2008

**Bc. Přemysl Cimbálek**

# VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací


# DIPLOMOVÁ PRÁCE

## Grafické zobrazení relací mezi počítači v Internetu


Obor: Telekomunikační a informační technika

Jméno studenta: Bc. Přemysl Cimbálek

Vedoucí práce: Ing. Dan Komosný, Ph.D.

# Prohlášení

Prohlašuji, že diplomovou práci na téma na téma "Grafické zobrazení relací mezi počítači v Internetu" jsem vypracoval samostatně pod vedením vedoucího diplomové práce, s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně, dne 20.5. 2008

# Poděkování

Vedoucímu semestrálního projektu Ing. Danovi Komosnému, Ph.D. za užitečnou metodickou pomoc a rady při zpracování bakalářské práce.

Ing. Radimu Burgetovi za velmi výraznou pomoc při zpracovaní dané problematiky v programovacím jazyku Java. Nechť je mu provolávána věčná sláva.

Kolegům, kteří se podíleli na zpracování jiných částí souvisejících s projektem IPTV, zejména pak Bc. Mojmíru Jelínkovi.

Největší poděkování však zasluhuje moje rodina, která mě svojí psychickou a finanční podporou umožnila se dopracovat až k diplomové práci a téměř ke konci studia. Rovněž tak přítelkyním a přátelům za nemalou podporu v krušných chvílích, mimo jiných jmenovitě Lence Cibulkové, Pavle Švástové, Václavu Burdějovi a Martinu Brejchovi.

Děkuji také účastníkům internetových diskuzí za konzultace některých dílčích problémů.

# Obsah

# Seznam obrázků

# 1. Úvod

Není třeba polemizovat o tom, že se nacházíme v době, kdy se internet stal součástí našeho každodenního života. I když v některých případech nepřímo. Nejde již o prosté stahování dat typu server-klient, ale i o vyměňování dat mezi klienty bez potřeby serveru (P2P sítě). Tato data nejsou již pouze určena převážně pro zábavu a zdroj informací, jak tomu bylo do značné míry dříve, ale jedná se například o vzájemně komunikující systémy (banky, úřady ...), nové typy zařízení (domácí spotřebiče, senzory apod.) a v neposlední řade také komfortní přenosy hlasu a obrazu. To ovšem zapříčiňuje rostoucí požadavky na přenosy velkého množství dat vyššími rychlostmi, výpočetní a paměťový výkon serverů, rozšíření adresace a potřeba vyšší spolehlivosti přenosových cest a zdrojů. Koncepce internetu, která vznikla před desítkami let a protokoly, které na ní byly nasazeny, již nedostačují, a proto vzniká v poslední době mnoho projektů, jež se zabývají novými technologiemi, které budou moci internet využívat efektivnějším způsobem.

Jedním ze způsobů jak zrychlit a zefektivnit aplikace typu klient-server je přenesení části nutného výpočetního výkonu a datového toku na servery, které budou pro hlavní server zajišťovat sloučení a popřípadě "předúpravu" dat od klientů. Takto vznikne hierarchická struktura, kdy máme pouze jeden server na vrcholu struktury, který vysílá či přijímá data dalším serverům v nižších vrstvách struktury a ty teprve komunikují s klienty.

## 1.1. Multicast IPTV Research Group

Jedním z projektů, zabývajích se právě vývojem nových technologií zejména pro multicast (výběrové všesměrové) vysílání televize pro TCP/IP sítě a tedy i Internet, je Multicast IPTV Research Group, projekt vzniklý na Ústavu telekomunikací Vysokého učení technického v Brně. Zabývá se zejména vývojem nového algoritmu, využití stavájících a vývojem nových protokolů a software, který by zajistil možnost vysílání interaktivní televize pro velmi velké množství uživatelů. Jedná se zejména o využití protokolu RTP (Real-time Transport Protocol), RTCP (Real-time Control Protocol) a principu hierarchické sumarizace (viz dále) .

Jak již bylo řečeno, televize IPTV slouží k přenosu digitálního televizního vysílání k uživateli po síti založené na rodině protokolů TCP/IP (resp. UDP). Z toho plyne, že nemusí být již jednosměrné, jako je tomu u klasického broadcast vysílání, ale může uživatelům poskytovat zpětnou vazbu ve formě interaktivity a na druhou stranu i sledování statistik, na co se uživatel zrovna dívá apod. Nevýhodou ovšem je, že přenosový kanál není dostatečně dimenzován na to, aby například na jednu internetovou ADSL přípojku bylo možné sledovat zároveň více kanálů. Určitým problémem je také doba přepínání kanálů, která je o poznání delší [22].

Princip vysílání spočíván v tom, že jeden program televize představuje multicast skupinu, do níž se uživatelé připojují. To umožňuje tarifikaci uživatelů dle doby připojení do skupiny a typu skupiny, možnost zvolit si jiný než právě vysílaný pořad (videotéka) apod.

## 1.2. Navazující součásti projektu

Díky rozsáhlosti problematiky je projekt rozdělen na více částí. Část vizualizace přímo navazuje či spolupracuje s následujícimi pracemi:

### 1.2.1. Zjištění fyzické pozice počítače v Internetu - Relovský Josef, Bc.

Zadáním je vytvořit v OS Linux skripty, které zjišťují časy odezev každého připojeného uzlu vůči zvolenému počtu referenčních bodů. Pro každý prvek se podle těchto časů vytváří poziční vektor, kde je znázorněno na prvním místě umístění referenčního bodu s nejnižší odezvou a na posledním s odezvou největší. Dále je v práci určeno z aktivních prvků počet pomocných prvků (Feedback Target - FT) i s jejich vektory. Vektory koncových uzlů se porovnávají mezi sebou s vektory vybraných pomocných prvků FT. Koncový uzel s nejshodnějším vektorem pomocného prvku se k tomuto prvku přiřadí. Podle těchto vektorů je vyhotoven tzv. „strom“, který znázorňuje relační vazby mezi uzly.

### 1.2.2. Sběr dat z velkého počtu počítačů pomocí hierarchické sumarizace - Jelínek Mojmír, Bc.

Hierarchická sumarizace dat, která byla realizována tímto projektem, obnáší sumarizaci došlých dat z koncových uzlů na podřazených serverech. Tzn. z došlých paketů jsou přečteny informace, tyto informace jsou ukládány do

paměti. Program cyklicky tyto data čte a vypočítává jejich průměrné hodnoty. Tyto hodnoty pak v jednom paketu posílá o úroveň výše, v našem případě na hlavní server.

## 1.3. O této práci

Úkolem je zobrazit relaci mezi počítači ve zmiňované hierarchické (stromové) struktuře, tzn. vykreslení pozice a popřípadě dalších informací o serverech a jim podřazených serverech a stanicích. Projekt je zaměřen na diskusi a volbu vhodné metody zobrazení na základě pochopení principu hierarchické sumarizace. Způsob zobrazení, volba metody, algoritmu a prostředků bude zdůvodněna v praktické části této práce, přičemž nelze postihnout všechny možné způsoby vizualizace relací, proto zde budou okrajově popsány pouze vybraná řešení s tím, že mnou použitá metoda či její možné modifikace je diskutována podrobněji.

Pro čtenářovo uvedení do problematiky je také nutné se zmínit o principech IPTV televizního streamovaného vysílání a jeho rozdílu oproti klasické televizi. Bude také nastíněna problematika protokolů využívaných pro vysílání a zpětnou vazbu v IPTV, včetně protokolu TTP (Tree Transmission Protocol), který vznikl pod projektem Multicast IPTV Research Group k zajištění signalizace na velmi rozsáhlých sítích. Projekt úzce spolupracuje s celosvětovou sítí PlanetLab, která slouží pro testování síťových aplikací, proto je v práci uveden princip a příklady projektů v této síti.

# 2. IPTV vs. "klasické" TV vysílání

U klasického analogového či digitálního televizního vysílání jsou televizní signály z televizního studia rozvedeny neveřejnou modulační sítí k pozemním vysílačům, které tvoří síť, jejímž úkolem je pokrýt určitou část území dostatečně silným signálem [28]. Velikost a vzhled vysílačů se liší dle potřebného výkonu a funkce, je možné do sítě zařadit i televizní opakovače. Princip samotného vysílání je založen na broadcast vysílání a je znázorněn na následujícím obrázku (klasické TV vysílání). Vysílání je jednosměrné pouze od vysílače k přijímači, přičemž jsou vysílány všechny kanály s tím, že až uživatel si na přijímači navolí, který kanál chce v danou chvíli přijímat. Je tedy zřejmé, že samotný vysílač nemá k dispozici žádné informace o tom, kolik přijímačů je na daný kanál připojeno. Výhodou ovšem je, že takto lze vysílat velmi velké množství kanálů (dle přidělených kmitočtových pásem).


**Obrázek 2.1. Klasické TV vysílání**

U digitálního pozemního i družicového televizního vysílání je možné v jednom frekvečním pásmu umístit více televizních programů. Nevýhodou ale stále zůstává to, že je v podstatě všem uživatelům vysílán stejný obsah, bez ohledu na požadavky a přání uživatele, mimo digitálních televizí (DVB-T, DVB-S, DVB-C), kde je určitá míra interaktivity zajištěna služebními informacemi, které jsou vysílány spolu s programovými kanály a starají se zejména o automatické řízení funkcí kodéru v přijímači. Umožňují se uživateli například orientovat v právě vysílaných službách. Účelem této práce není podrobně popisovat rozdíly mezi jednotlivými druhy televizního vysílání, nicméně je vhodné zmínit společný znak klasického digitálního vysílání a TV v IP sítích. Je jím většinou shodná metoda komprese obrazu a zvuku MPEG.

## 2.1. MPEG-2

Standardy MPEG jsou určeny pro kompresi datového toku zejména pohyblivých obrazů s podporou různých formátů videa a velikosti obrazu s přenosovou rychlostí od 1,5 Mbit/s. Tento formát je standardizován normou ISO/IEC IS 11172. Ta nespecifikují postup kódování (kodér se liší dle účelu), ale specifikuje bitový tok videa, sémantiku kódování. Vychází ze standardu JPEG, to znamená, že komprese se dosahuje za pomoci ztrátové konverze barevného prostoru, bezeztrátové diskrétní kosinové transformace DCT a entropickým kódováním VCL. Zavádí také snížení časové redundance odhadem pohybu a definuje tři typy snímků s různým zpracováním:

- Snímky I (Interframe Coded Frames) - obvykle se opakují po 12 snímcích, jsou zpracovány pomocí DCT bez DPCM, jsou tedy úplné a je k nim přímý přístup.
- Snímky P (Predicated Frames) - přenáší se pouze diference aktuálního snímku vůči předchozímu snímku P nebo I. Tato dopředná jednosměrná predikce snižuje bitovou rychlost dvakrát.
- Snímky B (Bidirectionally Coded Frames) - přenáší se diference aktuálního snímku B interpolací průměru předcházejícího a následujícího snímku I nebo P. Touto obousměrnou predikcí se snižuje bitová rychlost až osmkrát. Pro predikci musí být v paměti uloženy snímky, z nichž se předpovídá, a proto je nutné změnit jejich pořadí, proti pořadí při snímání.

Standard MPEG-2 se používá pro kompresi obrazových televizních dat s prokládaným řádkováním a je uzpůsoben pro bitovou rychlost 2 až 80 Mbit/s. Umožňuje pracovat v celosnímkovém i půlsnímkovém módu, který je vhodnější pro dynamicky proměnné obrazy.

MPEG-2 standard zahrnuje další velmi důležité vlastnosti, jako je například škálovatelnost - bitový tok je složen z více částí hierarchického kódování, tudíž je možné jej demultiplexovat na části s vyšší a nižší kvalitou obrazu [28]. Dekodér si tudíž může vybrat, jaká kvalita lépe vyhovuje jeho možnostem. Škálovatelnost může být v oblasti SNR (vyšší vrstvy poskytují více kvalitní video doplněním DCT koeficientů k základní vrstvě), prostorová (používaná pro souběžné vysílání, základní vrstva poskytuje základní prostorové rozlišení; vyšší vrstvy nesou informace potřebné k doplnění základní vrstvy pro vyšší rozlišení), časová (vyšší vrstvy doplňují základní vrstvy o mezisnímky) a dělení dat (do méně chybového kanálu se vysílají hlavní data, do chybovějšího kanálu se vysílají méně důležitá data (složky vyšších frekvencí).

Standard definuje také úrovně (Levels) pro různé kvality přenosu:

- Low Level (LL) - podporuje rozlišení do 352x288 do 30 snímků za sekundu. Maximální bitrate 4Mbit/s.
- Main Level (ML) - podporuje rozlišení do 720x576 do 30 snímků za sekundu. Maximální bitrate 15-20 Mbit/s.
- High 1440 Level (HL) - podporuje rozlišení do 1440x1088 do 60 snímků za sekundu. Maximální bitrate 60-80 Mbit/s.
- High Level - podporuje rozlišení do 1920x1088 do 60 snímků za sekundu. Maximální bitrate 80-100 Mbit/s.

### Systémová specifikace standardu MPEG-2

Paketované bitové toky jsou rozděleny na pakety PES (Packetized Elementary Stream), jenž mohou nést pouze jeden typ dat (audio, video, ...). To usnadňuje zpracování synchronizace obrazu a zvuku a přemísťování paketů v paměti. PES signály se ve výsledku mohou multiplexovat do programového nebo transportního toku. Programový tok má různě velké pakety a je výhodný pro bezeztrátové prostředí, kdežto transportní tok má pakety pevné délky 188 bytů (což usnadňuje synchronizaci) a je navržen pro prostředí s možným výskytem chyb. U sítí typu Ethernet má rámec maximální velikost 1518bytů [16] (dle typu, někdy se uvádí 1520bytů), což znamená, že v něm může být nejvýše 7 MPEG paketů. Délka datové části je tedy 7x188 = 1316bytů. Umožňuje také vložení jednoho nebo více programů v jednom streamu a mohou být pomocí něj přenášená i jiná data než MPEG-2, například MPEG-4, H.264 a podobně.

V neposlední řadě je nespornou výhodou standardu MPEG-2 možnost šifrování a schopnost řízení bitového toku prostřednictvím třetí strany.

## 2.2. MPEG-4

Tento standard, uvedený normou ISO/IEC 14496, byl vyvinut pro širokou škálu aplikací, pro datové toky o různých bitových rychlostech a video různé kvality. Jeho výhodou oproti MPEG-2 je vysoký stupeň komprese, zvládá přenosy po velmi pomalých linkách o minimálních rychlostech už 4,8 kbit/s, ale i kodování filmů o rychlosti 4Mbit/s.

Audiovizuální scéna je zde definována jako soubor objektů, které mezi sebou mají časovou a prostorovou vazbu, není to tedy pouhá sekvence obrázků se zvukem, i když obraz a zvuk se kóduje odděleně. Jako hlavní objekt pro standard je objekt obrazový, který má určitý tvar, texturu a pohyb. Z takovýchto objektů je tvořena samotná scéna, přičemž již nemusí být pravoúhlé, jak je tomu u MPEG-2. Každý je kódován samostatně, takže je zcela nezávislý na jeho okolí či pozadí scény. Objekty je možné slučovat do skupin. Vznikne tak hierarchická struktura obrazových objektů, například pohybující se člověk, statické pozadí, audio hlas člověka. S těmito objekty je možné ve scéně manipulovat, a to zejména je umístit na určité pozice v souřadnicovém systému, transformovat, sdružovat objekty, nahrazovat je jinými a popřípadě odstraňovat.

### 2.2.1. Kódování obrazu v MPEG-4

Základním stavebním prvkem obrazové časové sekvence je obrazová objektová rovina VOP (Video Object Plane), která je zakódována do několika oddělených objektových vrstev VOL (Video Objec Layer). Dekódováním všech těchto vrstev se obnoví původní zakódovaná videosekvence. Standard využívá jednotný algoritmus pro kódování jak pohybu, tak tvaru a struktury. Tento algoritmus je založen na hybridní DPCM. Podobnost se standardem MPEG-2 je i v typech obrazových objektových rovinách:

- I-VOP (Intra-Frame) – rovina je kódována bez predikce.

- P-VOP (Predicted) – rovina s predikcí.
- B-VOP (Bidirectionally) – rovina predikovaná z předešlé i následující roviny.

Každá VOP je rozdělena na makrobloky, které jsou po kódování 2D DCT transformací a koeficienty DCT kvantovány, následuje „cik-cak“ čtení koeficientů a entropické kódování.

Odhad pohybu a jeho kompenzace se provádí mezi bloky B-VOP a P-VOP podobně jako je tomu u MPEG-2, nicméně s tím rozdílem, že objekty mají libovolný tvar a je možné kompenzovat pohyb až čtyřmi vektory pohybu, které nemají hranice (vektor může pomyslně směřovat mimo obrazovou plochu). Zavádí se také globální kompenzace pohybu například pro pohyb kamery, zoom apod.

### 2.2.2. Přenos obrazu

Obraz pro paketový tok dat je přenášen v paketech podobných předchozímu MPEG-2, obsahuje resynchronizační značky, hlavičku, HEC a tyto pakety mohou být rozděleny na více částí - první část přenáší důležité informace pro rekonstrukci obrazu (DC koeficienty, pohybové vektory a informace o kódování) a druhá část obsahuje pouze zbývající AC koeficienty, DC koeficienty a resynchronizační značku. Bez této části se lze obejít.

MPEG-4 je rozdělen do množství profilů a vrstev, včetně profilu pro streaming, který umožňuje dynamickou změnu rozlišení a zvyšuje odolnost proti chybám. Důležitou vlastností je škálovatelnost. Dekodér si podobně jako u MPEG-2 může zvolit, jaká kvalita přenosu mu lépe vyhovuje. Škálovatelnost je opět prostorová (redukovaný obraz, nadvzorkování), časová (vkládání mezisnímků pro zvýšení snímkové rychlosti), podle složitosti a podle kvality.

## 2.3. Služby u IPTV

V této práci byly již jednotlivé služby, které IPTV nabízí, naznačeny. Je zde určitá paralela se službami, které nabízí v současné době u nás zaváděné digitální pozemní vysílání DVB-T. Služby jsou velmi podobné, nicméně jsou zde i určité odlišnosti.

### Television anywhere

Z principů televize vysílané po IP sítích, tedy internetu, je zřejmé, že takovouto televizi je možné sledovat odkudkoli na světě. V praxi většinou ale poskytovatel služby IPTV omezuje dostupnost služby pouze na určitou podsíť.

### EPG - Electronic Program Guide

Elektronický programový průvodce je nabízen i v DVB-T. Jedná se o televizní program v elektronické podobě, který je součástí vysílání a je většinou nabízen zdarma. Umožňuje tedy zobrazit všechny dostupné stanice a jejich televizní program včetně rozšířených informací, například podrobnějšího obsahu filmu. Umožňuje také automatické časové zapnutí nahrávání [14].

### Addressable advertising

Tato služba je výhodou spíše pro provozovatele televizního vysílání. Jedná se o to, že díky větším znalostem o divákovi v IPTV (IPTV umožňuje mechanismy identifikace příjemce a jeho chování) je provozovatel schopen měnit obsah reklam "na míru" a v neposlední řadě i sledovat, jak uživatel na tyto reklamy reaguje.

### Pay-per-view

Služba je zejména pro speciální placené pořady (sportovní přenosy, erotické vysílání), kdy je pořad většinou vysílán opakovaně a divák si za jeho dekódování a zhlédnutí zaplatí. Existuje zde také více možností, jak dalece může být služba nabízena - divák si může zaplatit buď pouze jedno vysílání nebo možnost nahrávání apod [30].

### Video On Demand (VOD)

Jedná se o službu, kdy je zákazníkovi nabízena videotéka poskytovatele služeb. Uživatel si může z pohodlí domova vybrat film, na který by se chtěl dívat a ten je mu následně přehrán. Toto je výhoda IPTV oproti DVB-T, kde síť není příliš dimenzována na to, aby se vysílal ke každému účastníkovi jiný kanál [30].

## Další služby

IPTV kromě zde vyjmenovaných hlavních služeb nabízí i jiné výhody oproti DVB-T, zejména možnost v podstatě nekonečného množství pořadů, možnost videorekordéru a vyšší stupeň interaktivity.

## 2.4. Princip IPTV

Jak již bylo řečeno, internetová televize IPTV slouží k přenosu digitálního televizního vysílání k uživateli po síti založené na rodině TCP/IP (resp. UDP). To s sebou nese výhody ale i nevýhody. V této části budou popsány základní principy IPTV vysílání televize.

### 2.4.1. Směry přenosu

Díky použití datové sítě již nemusí být vysílání jednosměrné, jak tomu je u klasického televizní broadcast vysílání, ale může uživatelům poskytovat zpětnou vazbu ve formě interaktivity a na druhou stranu i sledování účastníka a statistik, na co se uživatel zrovna dívá a podobně. Je tedy možné přesně identifikovat uživatele a jeho chování. Poskytovatel je tedy schopen zákazníkovi přizpůsobit televizní vysílání přesně dle jeho požadavků, a to včetně reklam a programové skladby [22].

Určitým problémem je také doba přepínání kanálů, která je o poznání delší. Oproti klasické televizi, kde je přepínání kanálů řešeno na úrovni přijímače, u IPTV je k uživateli vysílán pouze jeden program. O to, jaký je program k účastníkovi vysílán, se stará jiná část sítě (struktury IPTV) (viz dále v podkapitole Topologie IPTV). Přepínání programu může tak v horším případě trvat i několik sekund [22].

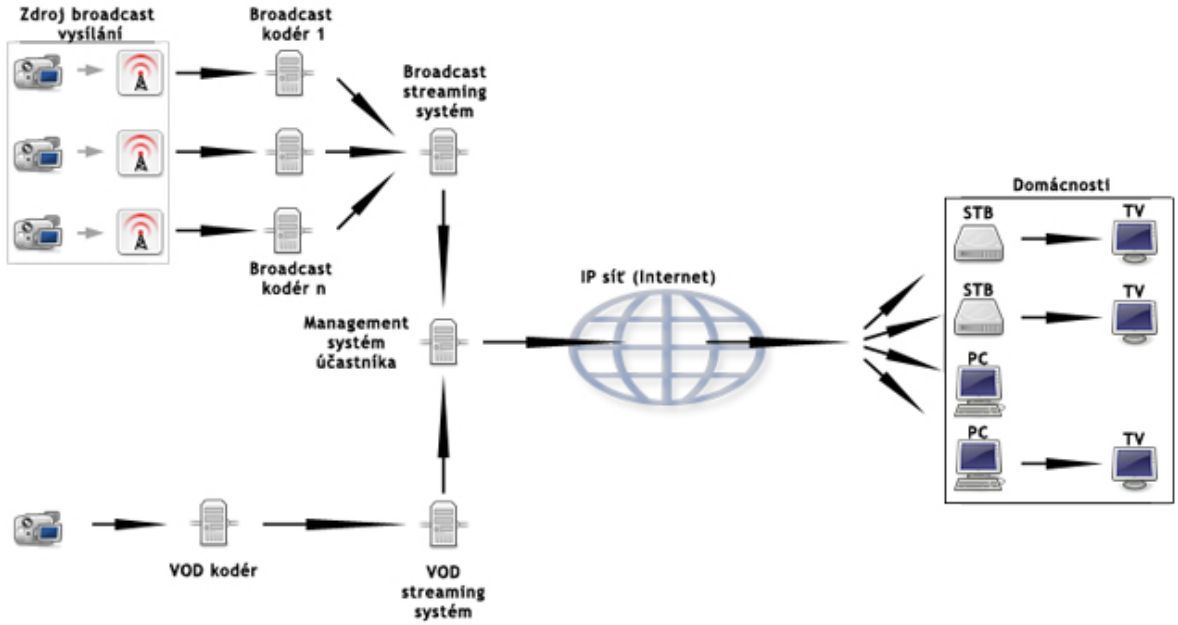
### 2.4.2. Přenosový kanál

Zásadní nevýhodou a odlišností oproti klasickému TV vysílání ovšem je, že přenosový kanál není dostatečně dimenzován na to, aby například na jednu internetovou ADSL přípojku bylo možné sledovat zároveň více kanálů. V literatuře [22] je uvedena modelová situace, kdy se každý ze členů jedné domácnosti chce dívat na jiný program. Přenosová kapacita dnešního účastnického vedení není taková, aby to bylo umožněno, lze přijímat pouze jeden kanál a ten rozvádět na více míst, ale nikdy více kanalů paralelně. Je zřejmé, že tento problém je spíše dočasného charakteru, jedná se o tzv. problém poslední míle. V dnešní době není nedostatek kapacity páteřních datových sítí. Jsou známy spolehlivé a cenově přístupné technologie pro tento typ sítí. Takových technologií je celá řada, většinou jsou založeny na optické technologii WDM a na ní nasazených protokolech jako ATM, SONET, SDH s kapacitami až 10Gb/s. Opačná situace je ale u sítí, které vedou přímo k zákazníkovi, tzv. přístupových sítí. Počet koncových uživatelů roste mnohem rychleji než zřizování nových přístupových bodů pro tyto uživatele. Budování přístupových sítí je totiž velmi drahé, i když existuje celá řada řešení.

Budování metalitických nebo optických sítí je vhodnou variantou především díky rychlosti a kapacitě takových sítí, ale je velmi náročné, většinou vyžaduje nákladnou fyzickou instalaci vedení přes veřejné i soukromé prostory. Pro poskytovatele je výhodnější využití stávajících sítí, již vybudovaných. Na zastaralé telefonní sítě pak nasadit vhodnou technologii, například ADSL, která dosahuje požadovaných parametrů pro náročné datové vysílání jakou IPTV bezpochyby je (záleží ovšem na vzdálenosti účastníka od ústředny). Vysílání IPTV o televizní kvalitě používající kompresi MPEG-2 potřebuje 3 až 3,5Mbit/s konstantního datového toku. K vysílání alespoň 3 programů na domácnost je tedy třeba připojení cca 10Mbit/s. Je možné také využít elektrorozvodné sítě. Alternativou k tomuto řešení jsou bezdrátové sítě, ať už provozované v licenčním pásmu (sítě FWA - Fixed Wireless Access) - bezdrátové sítě provozované v licenčním pásmu (u nás 3,5 GHz a 26 GHz, na buňkovém principu, s přenosy Point-to Multipoint, které nabízí vysokou kvalitu a spolehlivost) nebo různé druhy dvoubodového bezdrátového spojení a mobilní sítě. Posledně jmenované jsou ale pro zákazníka alespoň v našich lokacích finančně náročné a ani zdaleka nedosahují proklamovaných teoretických přenosových rychlostí (například HSDPA protokol pro UMTS by měl dosahovat maximální přenosové rychlosti až 14,4 Mbit/s, reálné rychlosti u poskytovatelů dosahují cca 1Mbit/s a nižší).

## 2.5. Topologie IPTV

IPTV je tvořeno dvěma hlavními vysílacími částmi: broadcast (všesměrové vysílání) a VOD (Video On Demand - video na vyžádání). Následující obrázek zobrazuje zjednodušené schéma topologie IPTV.




**Obrázek 2.2. Zjednodušené schéma topologie IPTV**

Systém je složen z následujících částí:

- Zdroj broadcast vysílání (Broadcast source). Zdroj (živého) všesměrového vysílání, většinou se jedná například o komerční kabelovou síť nebo klasické pozemní televizní vysílání.
- Broadcast kodér (Broadcast encoder). Do této části vstupuje většinou analogový signál nebo vysokorychlostní digitální signál z broadcast zdroje. Výstupem je datový tok (stream) formátovaný a komprimovaný pro přenos přes IP síť. Kodér je typicky řešený softwarově, ale může se jednat i o hardwarové řešení. Jak již bylo řečeno v předchozím, jedná se o kodéry MPEG-2 nebo MPEG-4.
- Broadcast streaming systém (Broadcast Streaming System). Většinou se jedná o server, který poskytuje jednotlivé kódované toky digitální televize pro velmi velké množství uživatelů. Server tedy může doručovat jak jednotlivé unicast (viz dále) nebo vícesměrové multicast (viz dále) vysílání pro více uživatelů, přičemž poměrně náročné je právě unicast vysílání, kdy server musí udržovat tísíce jednotlivých spojení. Pro představu je zde uváděno, že se jedná o jeden server, ve skutečnosti jde však o více serverů (sever farm), jednak z důvodů rozložení zátěže, jednak jako prevence proti chybám.
- Zdroj VOD a VOD kodér (VOD source a VOD encoder) - VOD (Video On Demand) neboli video na vyžádání. Zjednodušeně řečeno se jedná o službu, kdy místo toho, aby uživatel chodil do videopůjčovny, si pouze navolí v pohodlí domova film, na který se chce dívat a tento film je mu přehrán. Video data jsou tedy předem nahrána na médium (pevný disk, DVD apod.). VOD kodér se stará o převedení a formátování tohoto přednahraného vstupu na tok dat vhodný pro přenos přes IP síť. Většinou je tento převod řešen softwarově.
- VOD streaming systém (VOD Streaming System). Server, na kterém jsou uložena VOD data pro klienty. Je zřejmé, že aby byl server schopen obsloužit velmi velké množství klientů, musí uchovávat velké množství dat a ty musí být schopen vysílat. Opět se tedy v reálu jedná o skupinu více serverů (server farm), aby byla zajištěna maximální dostupnost služby a eliminovala se možnost chyb. Datové úložiště pro video data je většinou také rozloženo po síti kvůli prevenci vzniku chyb a výpadků.
- Management systém účastníka (Subscriber Management System). Stará se o obsluhu jednotlivých zákazníků a zajišťuje další zákaznické služby, jako je například EPG (Electronic Program Guide - on-line seznam s dostupnými programy a jejich obsahem) a tarifikace.
- IP síť. Paketová síť starající se o správné doručování a směrování dat. Není zde nutné vysvětlovat principy IP sítě, nicméně je nutno podotknout, že se v případě IPTV většinou jedná o síť postavenou na technologii ATM / DSL.

- STB (Set-top Box) nebo PC. Na účastníkově straně je třeba zařízení, které převede datový tok z vysílacích serverů na digitální nebo analogový signál, který může být reprezentován televizí nebo monitorem. Set-top Box může obdobně jako PC zajišťovat i přidané služby, jako je například webový prohlížeč, EPG a podobně.

- Televize nebo monitor. Služba IPTV je zaměřena na to, aby poskytovala pokud možno video v kvalitě Standart Definition Television Signal (SDTV), tedy obvyklý televizní signál v analogových televizních soustavách se standardy PAL/SECAM/NTSC na rozlišení většinou 720×576 resp 720×486 a poměrem stran 4:3. Se zavedením rychlejšího připojení pro koncové uživatele umožňuje samozřejmě vysílání televize v kvalitě HDTV (High Definition Television Signal), tedy v poměru stran 16:9 a rozlišení 1280×720 resp. 1920×1080. Tento formát předpokládá přístupové linky postavené na technologiích ADSL2+, VDSL nebo FTTH (Fiber to the Home).

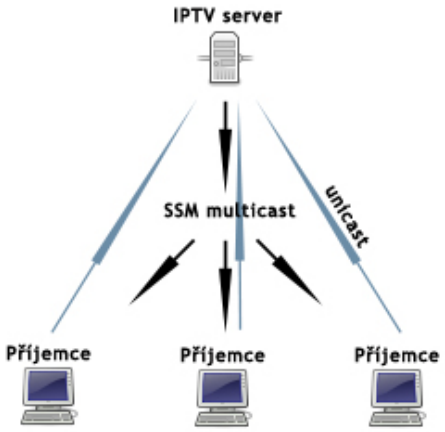
## 2.6. Multicast a unicast

Jak již bylo řečeno v předchozích podkapitolách, IPTV vysílání neumožňuje pouze jednosměrné vysílání ve směru k účastníkům, umožňuje také zpětnou vazbu od účastníků směrem k poskytovateli služby. Je zřejmé, že samotné vysílání video dat bude směrem k velmi velkému množství účastníků. Tito účastníci by se mohli rozdělit do jednotlivých skupin, z nichž každá sleduje jiný pořad. Z tohoto faktu a z prostředí IP sítě plyne základní předpoklad, že není možné data vysílat broadcast (souběžné vysílání ke všem příjemcům) způsobem, jak je tomu u běžného televizního vysílání. Využívá se proto jiného typu vysílání, tzv. multicast.

### 2.6.1. Unicast

Unicastový způsob vysílání je historicky starší než multicast a vychází z předpokladu IP protokolu, kdy komunikují pouze jeden zdroj dat a jeden příjemce. Z toho plyne, že tento typ vysílání nebude příliš vhodný k vysílání IPTV, pro každého účastníka by muselo být vytvořeno samostatné spojení, což je vzhledem k počtu účastníků nereálné.

Výhodné je ovšem využití unicast vysílání k přenosu signalizace a interaktivity koncových uživatelů. Tato komunikace totiž probíhá v podstatě v opačném směru než vysílání audio a video dat. Směřuje zpět od uživatele k vysílacímu serveru a zde je vhodné, aby probíhala pouze od účastníka k vysílacímu serveru (nebo sumarizačnímu serveru - viz dále) a ne všesměrově.




**Obrázek 2.3. Zjednodušený princip směrů toků dat v IPTV**

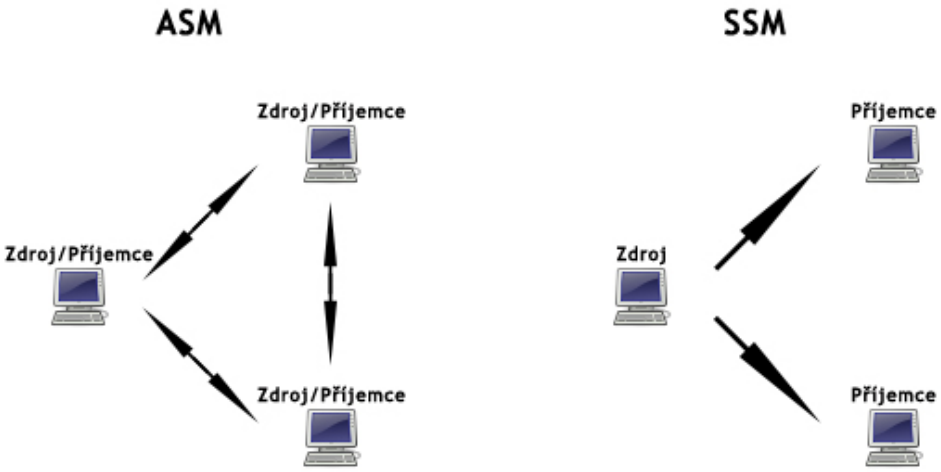
### 2.6.2. Základní vlastnosti multicastu

Technologie skupinového vysílání (multicast) slouží převážně k vysílání od jednoho zdroje k více příjemcům s tím, že všichni příjemci očekávají stejná data. Dobré vlastnosti má tento typ přenosu pro multimediální aplikace, kdy jeden vysílač posílá příjemcům stejná data v podstatě v reálném čase, přičemž vysílači je, dalo by se říci, jedno, kolik příjemců data zpracovává. O směrování v síti a distribuci směrem k příjemcům se starají směrovače,

vysílač totiž vysílá data na skupinovou adresu (třída D). Síťové prvky provádějí replikaci paketů a jejich hromadné rozesílání do požadovaných směrů, kde jsou zájemci o příjem dat [26].

### 2.6.3. Typy multicastu

Multicastový model vysílání nabízí dvě varianty, viz následující obrázek.




**Obrázek 2.4. Typy multicastu**

#### 2.6.3.1. ASM (Any Source Multicast)

V tomto typu vysílání síť vždy rozlišuje, kdo je zdrojem vysílání a všechna data od jednotlivých zdrojů vysílání jsou dopravena ke všem příjemcům. Je tedy zřejmé, že taková struktura je velmi složitá a ne příliš vhodná při velmi velkém počtu účastníků. Používá se tedy spíše pro on-line hry, audio a video konference, kde je relativně malé množství účastníků.

#### 2.6.3.2. SSM (Specific Source Multicast)

Tento způsob vysílání je do určité míry odvozen do Any Source Multicast, nicméně liší se v tom, že musí být přesně definován zdroj dat. Pokud se na síti vyskytne jiný zdroj dat vysílající na stejnou multicast adresu, tento druhý zdroj není již přijímán. Princip je takový, že vysílací server vysílá data na adresu multicastové skupiny. K vysílacímu serveru se uživatelé ale přihlašují výběrem na jeho unicast adresu [11].

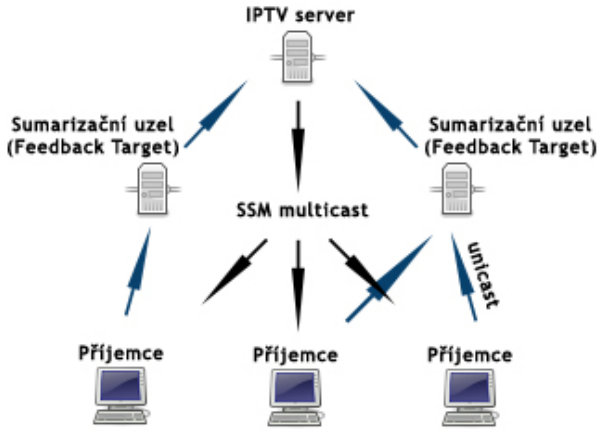
Pro vysílání IPTV je tedy z principu vhodné Source Specific Multicast vysílání, kdy je zdrojem dat zdroj streamu televize a uživatelé se pouze připojí ke správnému zdroji programu, který chtějí.

# 3. Princip hierarchické sumarizace

Součástí této práce je nastudovat princip hierarchické sumarizace, která je v řešení IPTV použita. Jak již bylo naznačeno v předchozí kapitole, řešení IPTV přenosu dat spočívá v tom, že streamovaná data jsou SSM multicastem vysílána ke skupinám příjemců dle toho, ke které skupině (vysílanému programu) se uživatelé připojili. Aby byla zajištěna interaktivita a možnost sledování chování koncového uživatele, musí být do struktury zaveden ještě zpětný kanál od účastníka k serveru, po kterém budou probíhat signalizační a informační data. Tento zpětný kanál je realizován unicast vysíláním, jelikož je třeba pouze komunikace koncový uživatel > server.

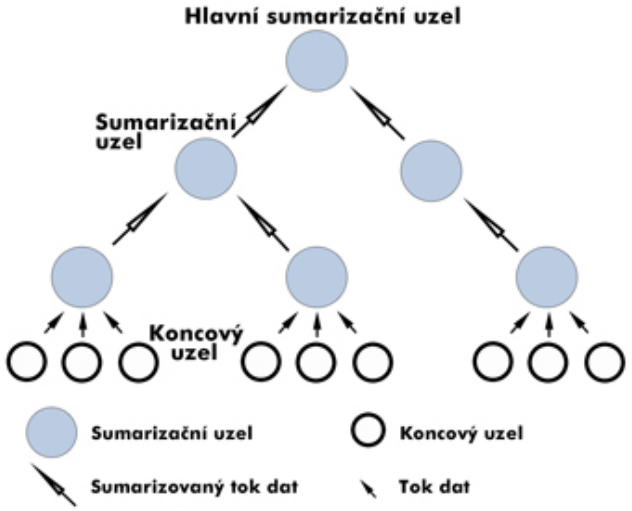
Signalizační a další data jsou postavena na protokolu RTP, RTCP (viz dále). Protokol RTCP (RTP) zajišťuje kontrolu nad multimediálními službami. Datový tok, který si může rezervovat je maximálně 5% celkového toku, což pro běžné provozy stačí. Problém nastává ovšem pro rozsáhlé streamovací systémy, kdy je na server připojeno velmi velké množství uživatelů, přičemž pro každého je třeba rezervovat určitý datový tok pro výměnu informací pomocí RTCP protokolu. Z toho je zřejmé, že pokud by byl zřízen od každého uživatele ke zdrojovému serveru zvláštní unicast kanál, kde bude vysíláno byť málo dat, dojde jednak k zahlcení linek a zejména pak k zahlcení samotného serveru tím, že bude zpracovávat a vyhodnocovat data od uživatelů.

Řešení tohoto problému spočívá v tom, že se uzly sítě rozdělí do dvou skupin: koncové uzly a sumarizační uzly [12]. Koncové uzly se chovají jako běžní uživatelé. Sumarizační uzly zavádí novou funkci, sbírají data z náležících koncových uzlů, ty seskupují a z dat vytváří sumarizační zprávy, které poté posílají nadřazenému sumarizačnímu nebo již kořenovému uzlu. Následující obrázek symbolicky popisuje vzniklou situaci zasílání signalizace unicast vysíláním přes sumarizační uzly.




**Obrázek 3.1. Schéma struktury se sumarizačními uzly**

Obrázek vystihuje pouze obecný princip a zcela nevystihuje princip sumarizačních uzlů respektive hierarchické agregace. Samotné zavedení několika sumarizačních uzlů opět nemusí postačovat pro velmi velké množství sumarizačních dat. Proto je sumarizace řešena hierarchicky ve stromové struktuře. Uzly tedy předávají sumarizovaná data nadřazeným uzlům na vyšší úrovni, které opět provedou vyhodnocení dat, jejich sumarizaci a data pošlou o úroveň výše. Tak se inforámce dostává až k hlavnímu uzlu na vrcholu hierarchické struktury, jak je naznačeno na následujícím obrázku.




**Obrázek 3.2. Hierarchická struktura sumarizačních uzlů**

Vlastní hierarchická topologie sumarizačních serverů různých úrovní není statická. Struktura stromu se mění dynamicky dle počtu připojených uzlů, jejich fyzické pozici v síti, vlastností přenosové sítě a potřeb provozovatele. Její výhodou je, že výstupem této struktury jsou informace "posbírané" v hlavním sumarizačním uzlu (Main Feedback Target nebo Root Feedback Target). Výhodou této struktury je, že na zpracování informací od velmi velkého počtu koncových uzlů se podílí celé struktuře rovnoměrně tak, aby byly sumarizační uzly náležitým způsobem vytíženy, nikoli přetíženy. Lze ji také využít nejen v řešení signalizace v IPTV ale obecně v jiných podobných systémech s velkým množstvím účastníků.

## 3.1. Využívané protokoly

V předchozí podkapitole byl rozebrán princip hierarchické sumarizace. K ucelení informací o problematice a osvětlení, jak signalizace a uspořádání hierarchické stromové struktury funguje, je nutno zmínit základní protokoly, které se na sumarizaci podílí.

### 3.1.1. RTP

RTP (Real-time Transport Protocol) protokol na transportní vrstvě slouží zejména k podpoře pro koncové multimediální přenosy v reálném čase. Nezaručuje tedy správné doručení dat ve správném pořadí paketů, jelikož nemůže ovlivnit momentální možnosti sítě. Jeho úkolem je spíše přenášet pořadová čísla paketů, aby si až koncová aplikace mohla sama rozhodnout, které pakety chybí, popřípadě pakety poskládat do správného pořadí. Nabízí také synchronizaci přenosu paketů a zjištění ztrátovosti.

RTP je jakousi obálkou pro transportní protokoly, které přenáší samotná data. Nejčastěji je využit UDP protokol (na portech 5004, 5005 a 6970), ale je zde možnost využít i jiné protokoly [25]. Je navržen jak pro jednosměrné, tak i pro obousměrné skupinové přenosy, z čehož plyne, že jej lze využít například ve videokonferencích, pro IP telefonii i pro přenos videa. Zajímavá je i možnost komprese při přenosu videa, a to jak datové, tak i části záhlaví.

Záhlaví RTP protokolu je tvořeno číslem paketu (sequence number) pro zjištění ztrát nebo duplicity paketů a označení typu obsahu (payload identification). Kódování obsahu se může změnit, pokud se má přizpůsobit rozdílům v šířce pásma. RTP pakety obsahují indikaci začátku a konce rámce (frame indication), identifikaci zdroje (source identification) a synchronizaci (intramedia synchronization) pro detekci různého kolísání zpoždění v rámci daného toku a pro potřebnou kompenzaci tohoto kolísání při vlastním přehrávání obrazů a zvuků. Formát RTP datagramu je jednoduchý a obecný, takže vyhovuje všem aplikacím v reálném čase (existuje pouze jeden typ zprávy RTP). Mezi typy dat RTP zprávy jsou G.721 audio, GSM audio, G.722 audio, MPEG audio, G.728 audio, H.261, MPEG-1 a MPEG-2 video [25]. Konkrétně v IPTV se RTP stará o přenos SSM multicastu.

### 3.1.2. RTCP

RTP protokol neumožňuje žádný mechanismus na zjištění doručení, včasného doručení paketů ani zjištění správného pořadí [25]. Proto je většinou protokol RTP nasazován ve spojení s protokolem RTCP, na UDP portu vždy o jeden vyšší než RTP.

RTCP (Real-time Transport Control Protocol) je řídicím protokolem pro přenos v reálném čase. Jeho úkolem je periodicky zasílat pakety od každého účastníka relace všem ostatním účastníkům za účelem monitoringu a řízení výkonnosti. Perioda zasílání je ovlivněna zejména počtem účastníků.

Protokol funguje oboustranně, zasílá pakety jak od zdroje k účastníkovi (SR paket), tak naopak od účastníka ke zdroji (RR paket). Tak je příjemce schopen na základě informací z RTCP paketu detekovat ztrátu paketů a provést kompenzaci zpoždění v síti a naopak vysílač může například dynamicky měnit rychlost vysílaného datového toku a kontrolovat zahlcení sítě.

Metody zajištění unicastového signalizačního kanálu jsou obecně dvě. Jednodušší je metoda přeposílání paketů ve zdroji, kdy zdroj přijímá všechny unicast RR pakety od jednotlivých příjemců a tyto pak přeposílá všem příjemcům pomocí multicastového vysílání. Výhodou je snadné rozpoznání zdroje paketů a jednoduchost metody, nicméně jsou zde určité mezery v neefektivitě metody, zejména ve velmi velkém množství nepotřebných dat putujících sítí. Například v IPTV není třeba příjemcům zpět přeposílat informace o zpoždění, které je důležité pouze pro zdroj vysílaných dat [11].

Druhá metoda vychází z předchozí, nicméně přijatá signalizační data v RR paketech slučuje podle různých matematických algoritmů. Jejich popis přesahuje rámec a účel této práce, nicméně výsledkem tohoto sloučení je značně komprimovaný tok dat. Nevýhodné je ovšem to, že tím pádem chybí přesné informace o jednotlivých uzlech zvlášť, k dispozici je pouze souhrnná informace. Tato metoda s sebou přináší nový typ paketu nazývaný XR (Extended Report). Paket obsahuje více specifické údaje než klasické RR pakety, mimo jiné například ztrátovost paketů, jitter, zpoždění a šířku pásma protokolu RTCP [11]. Pro účely IPTV může paket kromě jiného obsahovat informace o chování koncového uživatele, například požadavek přepnutí programu a podobně.

Jelikož je RTCP signalizační protokol, je snaha o jeho co nejmenší požadavky na datový tok. V literatuře [11] je uvedeno, že maximálně může RTCP obsadit 5% šířky pásma na jednoho uživatele s tím, že RR pakety obsazují 75% této šířky a SR 25%. Z toho je tedy zřejmé, že pokud budeme mít v síti velmi velké množství účastníků, aby bylo dosaženo maximální šířky pásma, bude perioda vysílání RTCP paketů velmi dlouhá a síť ztratí možnost flexibilně reagovat na změny nehledě na nepřijatelné zpoždění přenosu interaktivity od diváka k vysílacímu serveru a zpět. Proto jsou data sumarizována v hierarchické struktuře.

Komprimovaná data jsou mezi jednotlivými úrovněmi sumarizačního stromu zasílána jako tzv. RSI pakety, které vychází ze struktury RS paketů.

### 3.1.3. TTP

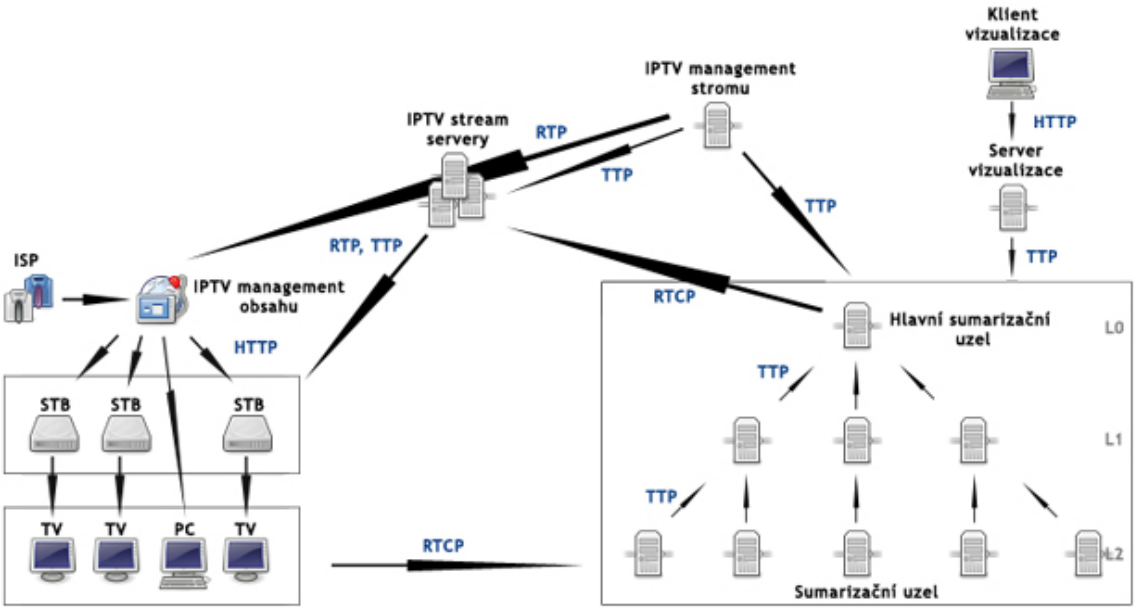
Byly zde rozebrány protokoly pro sběr dat, nicméně pro správné fungování dle předpokladů je nutno vytvořit onu stromovou strukturu. Je zřejmé, že bude nutné vytvořit vyvážený strom, kde koncové uzly musí být k sumarizačním uzlům připojeny tak, aby doba průchodu RTCP paketů v síti byla co možná nejmenší a zároveň byla struktura co nejlépe směrovatelná. Toto je řešeno algoritmem zvaným "binning", který hledá nejbližší uzly v síti dle zpoždění. Každému uzlu je přiřazen vektor a porovnáním těchto vektorů se koncový uzel rozhoduje, ke kterému sumarizačnímu uzlu se připojí. Problematika je zde uvedena velmi zjednodušeně, více informací je ve [20].

Dalším požadavkem na tvorbu hierarchické struktury je její dynamika, je nutný řídící uzel, který bude spravovat stromovou strukturu a bude jak sumarizačním, tak koncovým uzlům oznamovat, kam se mají do stromové struktury připojit. Jedná se o tzv. Feedback Transmission Manager (FTM). Tento uzel/sever přeskupuje strom dle aktuálních informací, které získal od všech ostatních uzlů a zasílá informace uzlům, kam se mají připojit. Rozhoduje tak zejména podle počtu uzlů spadajících pod jednotlivé sumarizační uzly a podle vzdálenosti uzlů.

K výměně informací pro FTM byl navržen protokol TTP (Tree Tnasmission Protocol). Tento protokol spolupracuje vedle RTP/RTCP a jeho obsahem je například indentifikátor stromu (FTID - Feedback Tree Identifier), jelikož stromová struktura může obsahovat více stromů, kdy některé uzly slouží pro více stromů. Paket TTP protokolu obsahuje také informace o jednotlivých sumarizačních uzlech, včetně volných systémových prostředků, průměrných velikostí odesílaných paketů, poloze ve stromu atp.

## 3.2. Struktura celé sítě pro přenos IPTV

Na následujícím obrázku je naznačena struktura celé sítě s využitím všech faktů uvedených v předchozích kapitolách. Je zde patrná jednak část pro samotné SSM vysílání streamovaných video dat, tak i napojení koncových uzlů do stromu pro zpětný přenos interaktivity a signalizačních dat. V obrázku jsou naznačeny i zmiňované nejdůležitější protokoly, které se podílí na přenosu dat.




**Obrázek 3.3. Sturktura IPTV v souvislosti s protokoly**

- IPTV management stromu (IPTV Tree Manager). Sever, který má za úkol koordinovat a zpracovávat informace o stromové struktuře sumarizačních uzlů a dynamicky reagovat na změny stavu. Se sumarizačními uzly komunikuje protokolem TTP a částečně se může podílet na správě vysílacího toku.
- IPTV stream servery. Převádí vstupní tok digitálních médií na tok vhodný pro přenos po IP síti. Pro Source Specific Multicast vysílání používají protokol RTP.
- IPTV management obsahu (IPTV Content Manager). Slouží ke správě obsahu IPTV, jako je například reklama, vlastní video stream, audio a podobně.
- ISP (Internet Service Provider). Poskytovatel internetových služeb i služby IPTV. Prostřednictvím IPTV Content Manager má přístup a dohled nad celou sítí.
- STB (Set-top Box). Převádí signál do podoby vhodné pro koncové zařízení, tedy například na analogový signál pro TV přijímač. Většinou obsahuje službu EPG - elektronický programový průvodce.
- Sumarizační uzel a hlavní sumarizační uzel (Feedback Target a Main Feedback Target). Jak již bylo řečeno výše, uzly, které mají za úkol zpracování a sumarizaci dat, která proudí od koncových uživatelů. Ostatní části systému tudíž komunikují většinou již pouze s hlavním sumarizačním uzlem (Main Feedback Target).
- Monitoring server. Ve struktuře je zařazen server, jehož úkolem je komunikovat s každou částí systému a získávat od ní data pro dohled, například o rychlosti spojení, zpoždění apod.
- Server vizualizace a klient vizualizace (Tree Visualization Server a Client). Úkolem serveru je shromažďovat data o stromu tak, aby se dala dynamicky vykreslit aktuální struktura stromu a podrobnější informace o uzlech.
- RTP/RTCP, TTP, HTTP. V schématu jsou zakresleny pomyslné "toky" jednotlivých protokolů.

Části Visualization Server a Client jsou hlavním tématem této práce jako celku. Otázkou tedy je, jak bude vypadat hierarchická struktura sumarizačních serverů (Feedback Target) a příjemců (Receiver). Navíc je třeba mít pře-

hledné informace o jednotlivých uzlech sítě pro vyhodnocení vytížení sítě, logické polohy jednotlivých prvků sítě a jejich spolehlivosti. Tyto informace s sebou nesou RTCP pakety, nicméně je třeba vhodným způsobem "vytáhnout" a zobrazit. Další části této práce budou tedy zaměřeny na diskusi vhodné metody zobrazení na základě zde uvedených principů, struktury IPTV a hierarchické sumarizace. Jak již bylo řečeno, Main Feedback Target musí dynamicky reagovat na změny v síti a dle těchto změn přizpůsobovat stromovou strukturu sumarizačních serverů. Podobně je tomu s vizualizací dat, která je přímo navázána na onu strukturu.

# 4. PlanetLab

Se současným překotným vývojem internetu vyvstávají na povrch některé problémy, které je třeba řešit. Jsou to například rostoucí požadavky na datový přenos, výkon serverů a hlavně zastaralé protokoly, na kterých je internet postaven. Proto vyniká celá řada projektů, které si jako svůj úkol vytyčily zpracování takových nových technologií, aby vyhovovaly novým požadavkům na celosvětovou počítačovou síť.

Jedním z nich je i PlanetLab, který vznikl v roce 2002, nejprve jako spolek amerických univerzit (University of California at Berkeley, Princeton University a University of Washington). Později se k projektu připojily další univerzity a významné firmy, které pracují v oblasti IT, jako například Intel, Hewlett Packard, Bell Labs a Google. Dnes jsou součástí i nadnárodní organice zajišťující samotný provoz internetu a datových spojů, například AT&T, France Telecom apod [22].

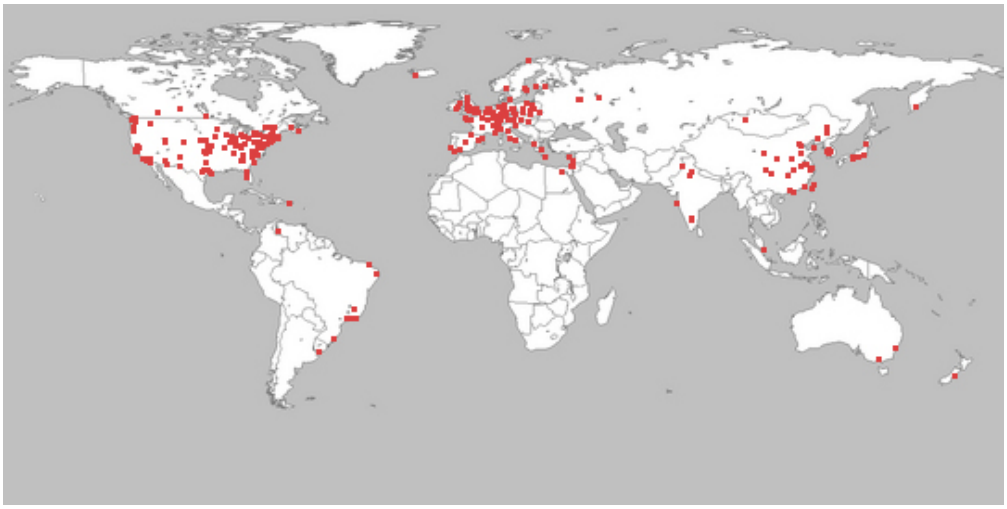
## 4.1. Souvislost této práce s laboratorní sítí PlanetLab

Obecný úvod o laboratoři PlanetLab je v této práci uveden z toho důvodu, že mnou provedená vizualizace relací mezi počítači bude prakticky otestována právě na síti PlanetLab. Jedná se o to, že v jiné části tohoto projektu je třeba vytvořit hierarchickou strukturu, jež simuluje reálnou síť, na které je vyvíjená technologie IPTV Hierarchical agregation testována. K tomu slouží jednotlivé uzly sítě PlanetLab. Uměle je určeno, které uzly slouží jako sumarizační a které se stanou pouze příjemci. Rozhodující pro tuto strukturu bude také zřejmě doba odezvy mezi jednotlivými uzly, kterou měří známá funkce "ping". A právě v této fázi je velmi těžké nasimulovat takovou síť a není zcela jisté, zda-li bude simulace odpovídat realitě. Proto je využito výhod sítě PlanetLab, kde lze algoritmy strukturalizace i vykreslení otestovat na skutečném prostředí a chování v síti. Výsledná struktura je popsána relační databází, kde jsou obsažena data o uzlech (viz dále).

## 4.2. Princip PlanetLab

PlanetLab je jedna velká laboratoř, sloužící pro výzkum nových síťových aplikací a technologií, složená z cca 700 nezávislých uzlů umístěných v různých koutech světa, včetně České republiky (3 uzly: "planetlab1.fit.vutbr.cz", "planetlab1.cesnet.cz" a "planetlab2.cesnet.cz"). Žádná firma sama o sobě nemá takové prostředky na vytvoření takovéto celosvětové sítě. Přispěním všech účastníků PlanetLabu (dnes cca 1000 institucí a firem) vznikla síť s uzly, které jsou dle určitých pravidel k dispozici všem uživatelům PlanetLabu k testování různých aplikací a protokolů tak, aby bylo zajištěno co nejvíce reálné prostředí. Na každém uzlu tedy může běžet spousta nezávislých aplikací najednou, jsou ovšem limitovány hardwarem a operačním systémem běžícím na uzlech, kterým je Linux Fedora Core 4.

Doposud zde byla řeč o jednotlivých uzlech, již ne však o serverech a klientech. Tato síť je tedy postavena na principu P2P (peer-to-peer) sítě, kdy se výměna dat neděje způsobem, jaký byl navržen před desítkami let, tedy že klient požádá o data server, nýbrž data se vyměňují v podstatě mezi decentralizovanými klienty navzájem, bez potřeby serveru, ovšem s tím, že ten, kdo data žádá, musí si je nejprve vyhledat. Flexibilita takovéto sítě spolu s dalšími výhodami vedla k tomu, že PlanetLab se stala prototypem pro tzv. Internet2, tedy "nová verze" internetu. Nebude se zřejmě jednat o úplně jiné a nové aplikace, ale spíše o takové, které budou fungovat na stávajících systémech, ale přinesou změnu, bude se tedy jednat o "nadstavbu" a "překrytí" stávajícího internetu.



**Obrázek 4.1. Uzly sítě PlanetLab (obrázek z http://www.planet-lab.org/node)**

## 4.3. Vybrané projekty na PlanetLab

Projekt PlanetLab se v současné době soustřeďuje na několik hlavních projektů. Snaha je především o to, aby každý uživatel internetu s sebou nemusel nosit přenosný počítač, ale veškeré aplikace a dokumenty byly dostupné přímo na síti. Zároveň je snaha vytvořit síť odolnou vůči virům a červům.

- CoDeeN - jedná se o systém, který v případě požadavku na nějaký soubor od mnoha uživatelů vytvoří kopie souboru tak, aby těchto kopií nebylo ani moc (zahlcené servery), ale ani málo (dlouhá čekací doba uživatelů). Takto to může být například u často navštěvovaných webových stránek, které se "replikují" na jiné uzly. S projektem CoDeen souvisí i podobné projekty, které se zabývají většinou aplikacemi na správu a monitoring PlanetLab, jako je CoTop a CoVisualize.

- OceanStore - systém na globální uchovávání dat. Data uživatelů budou rozdělena na fragmenty, které jsou posouvány sítí z místa na místo. Taková data jsou potom v podstatě nezničitelná, jelikož pokud některý uzel v síti vypadne, najde se vždy kopie fragmentu někde jinde.

- Scriptroute Network Measurement – nabízí celou řadu měřících nástrojů jak pro uživatele, tak pro developery, k měření odezev v síti, mapy routerů a podobně.

Kromě těchto projektů, které se často uvádí v odborných kruzích, jako zdrojové, jsou v PlanetLabu stovky dalších, které se jednak pokouší vyvíjet nové síťové technologie, ale také například slouží pro monitoring a práci s uzly v PlanetLab . Síť také slouží jako hosting pro webové stránky celé řady univerzit a společností.

# 5. Možnosti vizualizace hierarchické struktury z hlediska technologií

Jak již bylo řečeno, tato práce je zaměřena na zobrazení struktury a informací o sumarizačních uzlech. Existuje celá řada způsobů a řešení, jak data získávat a zobrazovat. Je možné například data vykreslovat již přímo v programu/skriptu, který bude získávat data o uzlech v síti, je také možné data uchovávat v nějakém formátu a ta dodatečně vyobrazovat. Je také nutné uvažovat, zda-li bude vhodné data uchovávat, kvůli možnosti zpětných výpisů, statistik a zobrazení, či bude třeba vykreslovat pouze aktuální situaci, čímž se nám množství dat zmenší pouze na popis aktuálního stavu.

Na základě předchozích uvedených faktech o struktuře IPTV a hierarchické sumarizaci budeme při návrhu řešení a tvorbě správné metody vycházet z následujících předpokladů:

- Dostupnost vizualizace. Vzhledem k tomu, že je IPTV technologií pracující na IP sítích, a tedy převážně na internetu, bude vhodné, když bude vizualizace dostupná také takto na veřejném médiu. Jedním z hlavních důvodů je přenositelnost, ten, kdo si data o stromu zobrazuje (dále jen "uživatel"), přistupuje k vizualizaci odkudkoli na světě všude tam, kde je připojení k internetu a aplikace internetového prohlížeče, a to pokud možno bez potřeby instalace dalšího software.

- Dynamika vizualizace. Tato problematika byla již zmíněna. Jedná se zejména o to, že uspořádání a počet sumarizačních uzlů se může v čase měnit dle potřeb sítě. Na toto by měla vizualizace také reagovat.

- Otevřenost. Je zřejmé, že vstupní data a potřeby pro vizualizaci se mohou měnit. Příkladem může být nastávající využití IPv6 adres místo stávajících IPv4. Je tedy vhodné, aby systém byl jednak na tyto změny připraven a jednak, aby jakákoli modifikace systému byla možná a jednoduchá.

- Variabilita. Z principu vysílání IPTV plyne, že pro jeden sumarizační uzel může být velké množství koncových uzlů (v řádu tisíců) a celkový počet účastníků může být i v řádu statisíců, na tento fakt musí být vizualizace připravena.

- Předpokládané hodnoty. Jak již bylo řečeno, předpokladem je, že nebude zobrazen pouze prostý strom rozdělení uzlů, ale bylo by vhodné zobrazovat i další informace o uzlech. Mezi tyto informace patří:

    - IP adresa v obou verzích (IPv4 i IPv6).

    - Port, na kterém probíhá spojení.

    - Příslušnost ke stromu. Jak již bylo řečeno, v systému může být více stromů.

    - Počet sdílení mezi stromy.

    - Velikost skupiny, do které daný uzel patří.

    - Velikost dat, které si mezi sebou dané sumarizační uzly přeposílají.

    - Perioda/interval, v jakém se posílají RTCP pakety.

    - Čas aktivace uzlu v síti.

    - Ztrátovost dat u koncových přijímačů.

    - Zpoždění dat (pouze pro koncové uzly).

    - Jitter (opět pouze u koncových uzlů). Jitter je kolísání zpoždění oproti dané referenční hodnotě. Tento faktor může mít velmi nepříznivý vliv na aplikace v reálném čase.

## 5.1. Datové úložiště

Pokud budeme brát v úvahu, že zobrazujeme informace o uzlech, bude zřejmě vhodné data uchovávat i pro pozdější zpracování, aby bylo možné zobrazovat, jak vypadala síť například před hodinou, dvěma dny apod. Budeme je tedy nějakým způsobem ukládat k pozdějšímu čtení, informace nebudou pouze proměnné (pole, struktury) v programu. To má ovšem poměrně značnou nevýhodu, množství dat bude velmi rychle narůstat. Dejme tomu, že máme například 10.000 uzlů, přičemž údaje o uzlech se ukládají každých 60 sekund. Interval ukládání je i tak kompromisem mezi potřebnou aktuálností zobrazení a náročností sběru dat o uzlech a jejich zpracování. Budeme tedy za den mít 14.400.000 "informací", za rok je to 5.256.000.000 informací, což je velmi vysoké číslo. Vezmeme-li v úvahu, že informace o uzlu je soubor několika hodnot, například 11, přičemž každá hodnota je reprezentována průměrně 3 znaky, máme celkem 173,448 . $10^9$ znaků ročně. Pro zjednodušení můžeme vzít v úvahu, že jeden znak je reprezentován jedním byte. Velikost samotných dat, a to bez režie, je 173GB. V dnešních poměrech to není nijak závratné číslo, ale je nutno také vzít v úvahu režijní data a fakt, že se bude s těmito daty čase pracovat, a to pokud možno v co nejkratším čase.

### 5.1.1. Zápis textového řetězce do souboru

Zvolíme-li naprosto jednoduchý způsob ukládání dat do souboru, například v obyčejné textové podobě na řádky textového souboru s jednotlivými položkami oddělenými oddělovacím znakem (čárka, středník), tedy ve formátu CSV, je poměrně problematické, jak data dodatečně vyhledávat. Procházení textových řetězců v tomto souboru nebude nikterak složité, ale vzhledem k počtu záznamů bude velmi časově náročné a velmi špatně čitelné z hlediska vyhledávání. Nevýhodou může být již zmiňovaná velikost takovéhoto souboru, a pokud jej budeme komprimovat, ztratíme komprimací a následnou dekomprimací kvůli zobrazení značné množství času. Výhodou je snadná propojitelnost s tabulkovými editory (Microsoft Excel, OpenOffice Calc)

### 5.1.2. Zápis do XML

Jedná se o vhodnější způsob, než je předchozí z hlediska lidské i strojové čitelnosti. Můžeme data ukládat do tagů, podobně, jako je tomu u HTML, tedy například (velmi zjednodušeno):

```
<zaznam id="200">
<ip_uzlu typ="ip_32">127.0.0.1</IP>
<port>6003</port>
<feedback_interval>5</feedback_interval>
<cas_aktivace>2008-05-01 22:20</cas_aktivace>
</zaznam>
<zaznam id="201">
<ip_uzlu typ="ip_32">192.168.0.1</IP>
<port>6003</port>
<feedback_interval>4.5</feedback_interval>
<cas_aktivace>2008-05-03 20:0<5/cas_aktivace>
</zaznam>
...
```

Takováto strukturovaná data je možné převádět do jiných formátů (PDF, RTF, HTML apod) a jinak zpracovávat například pro zobrazení, což je nespornou výhodou. Problémem je opět ale množství dat, tedy velikost souboru, který nám při postupném zápisu vznikne. Složitější je také optimalizace pro vyhledávací a zobrazovací algoritmus, i když u programovacích jazyků existují knihovny a metody, jak zacházet s XML daty (tzv. parser).

### 5.1.3. Databáze

Tento způsob se jeví jako nejvhodnější pro náš účel. Data jsou ukládána přehledně do tabulek, přičemž hlavní výhodou je, že o jejich zpracování a optimalizaci se z velké části postará sama databáze. Největší výhodou je rychlost a přehlednost. Byl vybrán typ databáze MySQL, pro její jednoduchost, snadnost správy a instalace a také pro snadnost implementace do webového prostředí.

Při návrhu databáze je nutný výběr kódování pro znaky. Předpokládáme data bez české diakritiky, nicméně pro případ, že by byla data s diakritikou vkládána je zvoleno kódování UTF8, které poskytuje univerzální prostředí pro zápis všech národních abeced. K zápisu využívá Unicode zápis znaků s tím, že je k dipozici 31 bitů pro zápis znaku. Výhodou je snadná konverze do kódování UCS-2 či UCS-4, které používají pro zápis textů moderní platformy. Mezi nevýhody patří obtížnější manipulace s textem, jako je převod mezi malými a velkými písmeny apod [3].

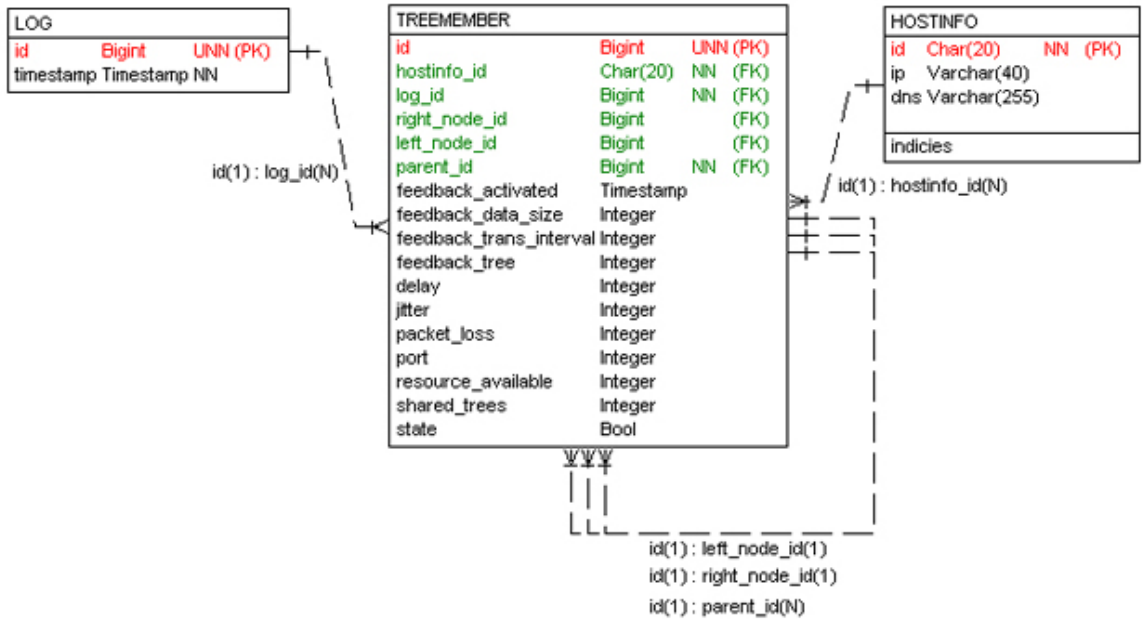
Pro správnou funkci databáze je nutné zvolit správný typ tabulek. MySQL totiž nabízí několik takových typů.

Typ MyISAM je dnes zřejmě nejpoužívanější úložiště dat. Je založeno na starším úložišti ISAM, které mělo v primárním souboru uložena v pořadí, v jakém byla do databáze zapsána. Dnešní MyISAM je ale narozdíl od svého předchůdce velmi rychlé a je optimalizováno pro dotazy typu SELECT, tedy výběry z tabulky. Nevýhodou je částečná fragmentace dat při ukládání dynamických tabulek (datové typy s proměnnou délkou). Engine také nepodporuje transakce, což jsou sady dotazů, které se musí vykonat zároveň a jejich přerušení nesmí ovlivnit tabulku. Transakce tedy musí být dokončena úplně.

Typ InnoDB používají transakce a jsou optimalizovány pro větší rychlost vkládacích dotazů INSERT a UPDATE. Optimalizována je také pro práci s velkým objemem dat.

I když předpokládáme poměrně velký objem dat, byl zvolen typ tabulek MyISAM, zejména kvůli jeho optimalizaci pro dotazy SELECT. Při vizualizaci totiž budou data vybírána tímto způsobem a počet výběrů dat zřejmě několikanásobně převýší počet vkládání dat.

Volba sloupců tabulky a jejich datový typ byl opět prováděn s ohledem na rychlost a co největší úsporu dat, abychom předešli zbytečné fragmentaci a nadbytečnosti dat resp. datových typů.




**Obrázek 5.1. Struktura databáze pro ukládání dat**

Jak již bylo řečeno, uložení dat do databáze je použito jednak z důvodu rychlosti jejich opětovného získání a zobrazování a jednak z důvodu zpětných výpisů. Pokud budeme data o stavu stromu ukládat v časových intervalech, budeme mít přehled o časovém vývoji stromu. Z povahy dat určených k zobrazení plyne, že můžeme využít relační databáze:

Existují tedy dva druhy dat o uzlech.

- "Statická" data. Samotná data o jednotlivých koncových uzlech připojovaných k IPTV vysílání a uzly ve stromu hierarchické sumarizace se budou měnit relativně pomalu, resp. předpokládáme, že například IP adresa a DNS jméno uzlů bude s časem stejné. Pro přehlednost a rychlost databáze je tedy můžeme ukládat do zvláštní tabulky.

- "Dynamická" data v tom smyslu, že abychom mohli bližší data o jednotlivých uzlech v čase sledovat, bude nutné tato data opakovaně do databáze ukládat. Jedná se zejména o proměnlivou strukturu stromu (měnící se nadřazený uzel a sousední uzly) a další informace, jako je například jitter, jejichž hodnoty budou pro každý záznam o stejném uzlu také různé.

Jednak kvůli přehlednosti databáze, jednak také z hlediska svázání databáze s vývojovým prostředím, kde budou data zpracovávána byla navržena struktura s několika tabulkami (viz předchozí obrázek), mezi sebou svázanými pomocí relací.

- TREEMEMBER - V této tabulce jsou uchovávána všechna data, která se s časem mění, tedy jednotlivé sousedící uzly a doplňující informace. Do této tabulky budou data o jednotlivých uzlech periodicky ukládána, záznamy o každém uzlu se tedy budou opakovat. Přes cizí klíče je tabulka v relaci jednak sama se sebou (left_node_id, right_node_id v relaci 1:1 s id) a s ostatními:

- HOSTINFO - Jak již bylo řečeno, IP adresa uzlu je v podstatě neměnná, proto jsou taková data uložena ve zvláštní tabulce s vazbou na tabulku TREEMEMBER přes id. Z toho také plyne, že více záznamů z tabulky TREEMEMBER může směřovat na jeden záznam z HOSTINFO.

- LOG - V tabulce je jednoznačně určen čas záznamu, který bude nutný pro zpětné vykreslování. Znamená to tedy, že pro jeden čas může být zaznamenáno více záznamů v tabulce TREEMEMBER, z opačné strany zjednodušeně řečeno: každý záznam v tabulce TREEMEMBER je jednoznačně časově určen pomocí záznamu v tabulce LOG.

- FT | RECEIVER - Určuje typ uzlu.

Na základě informací v tabulkách jsme tedy schopni sestavit podobu stromu pro vykreslení i s požadovanými doplňujícími informacemi a provádět i časově zpětné vykreslení. Význam jednotlivých atributů (sloupců) v databázi je následující:

- id BIGINT NOT NULL AUTO INCREMENT PRIMARY KEY - primární klíč číselného typu, který se automaticky inkrementuje a slouží k jednoznačné identifikaci záznamu dat o uzlu a propojení pomocí klíčů z ostatních tabulek.

- ip VARCHAR(40) NOT NULL - IP adresa uzlu. Pokud bychom předpokládali pouze IP adresy verze 4, tedy stávající 32 bitové adresy, mohla by být data reprezentována číslem BIGINT (např. 192.168.2.1 -> 192168002001). Znakový typ VARCHAR je volen proto, že je počítáno i s budoucím zápisem IP adresy verze 6 (např. 2001:0db8:85a3:08d3:1319:8a2e:0370:7348)

- dns VARCHAR(255) NULL - DNS jméno uzlu. Jelikož bude záznam použit pouze pro sumarizační pakety (koncoví uživatelé zřejmě nebudou mít DNS jméno), může být záznam nulový.

- activated TIMESTAMP - časový otisk doby, kdy byl uzel v síti aktivován, nebo-li kdy se do sítě připojil.

- state BOOL NOT NULL (neboli také TINYINT(1) ) DEFAULT '1' - nenulová logická hodnota. Použije se v případě, že je známo, že je již uzel ze struktury z nějakého důvodu vyřazen, jinak je vždy nastaveno na logickou hodnotu true (1).

- indicies - indexy nad tabulkou HOSTINFO, konkrétně nad poli id a ip, slouží zejména k rychlejšímu vyhledávání.

- port INT(5) NULL - záznam o portu, na kterém probíhá komunikace.

- type VARCHAR(20) NOT NULL - typ uzlu. Do tohoto sloupce budou ukládány hodnoty "feedback_target" nebo "receiver" pro sumarizační resp. koncový uzel.

- feedback_tree INT NOT NULL - číslo stromu, do kterého uzel patří.

- resources_available INT NULL - volné systémové prostředky, týká se zejména sumarizačního uzlu.

- shared_trees INT NULL - počet stromů, ve kterých uzel vystupuje

- feedback_data_size - velikost paketu se sumarizačními daty, týká se opět pouze sumarizačního uzlu.

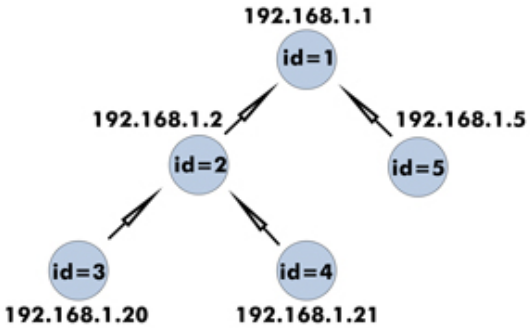
- feedback_trans_interval INT NULL - doba mezi odesláním dvou sumarizačních paketů.

- packet_loss INT(3) NULL - ztrátovost paketů v procentech.

- delay INT NULL - zpoždění paketů. Tento údaj se vkládá pouze u koncových uzlů.
- jitter INT NULL - kolísání zpoždění v síti oproti referenční hodnotě. Opět se předpokládá pouze u koncových uzlů.
- timestamp TIMESTAMP NOT NULL - Toto pole slouží k jednoznačné časové identifikaci záznamu kvůli zpětným výpisům.

## Ukázka dat v databázi

Na následujících obrázcích je uvedena ilustrační modelová situace a její zápis v databázi.




**Obrázek 5.2. Model stromu**

| id | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| delay | 1249 | 1370 | 532 | 532 | 532 |
| feedback_activated | 2008-05-20 20:37:12 | 2008-05-20 12:05:11 | 2008-05-20 12:05:13 | 2008-05-20 12:05:13 | 2008-05-20 12:05:13 |
| feedback_data_size | 3616 | 8903 | 4803 | 4803 | 4803 |
| feedback_trans_interval | 5 | 5 | 5 | 5 | 5 |
| feedback_tree | 1 | 1 | 1 | 1 | 1 |
| jitter | 4 | 16 | 16 | 16 | 16 |
| packet_loss | 52 | 9 | 9 | 9 | 9 |
| port | 6300 | 6300 | 6300 | 6300 | 6300 |
| resources_available | 73 | 94 | 94 | 94 | 94 |
| shared_trees | 1 | 2 | 2 | 2 | 2 |
| state | ☐ | ☐ | ☐ | ☐ | ☐ |
| type | feedback_target | feedback_target | feedback_target | feedback_target | feedback_target |
| x | NULL | NULL | NULL | NULL | NULL |
| y | NULL | NULL | NULL | NULL | NULL |
| leftNode_id | NULL | NULL | NULL | 3 | 2 |
| log_id | 1 | 1 | 1 | 1 | 1 |
| hostInfo_id | 1 | 2 | 3 | 4 | 5 |
| parent_id | NULL | 1 | 2 | 2 | 1 |
| rightNode_id | NULL | 5 | 4 | NULL | NULL |

**Obrázek 5.3. Ukázka databáze (tabulka TREEMEMBER) s příslušnými daty**

| id | address_dns | address_ip |
|---|---|---|
| 1 | vmware.moja2.cz | 192.168.1.1 |
| 2 | uzel2.cz | 192.168.1.2 |
| 3 | uzel3.cz | 192.168.1.20 |
| 4 | uzel4.cz | 192.168.1.21 |
| 5 | uzel3.cz | 192.168.1.3 |

**Obrázek 5.4. Tabulka HOSTINFO s příslušnými daty**

| id | timestamp |
|---|---|
| 1 | 2008-05-10 20:38:20 |
| 2 | 2008-05-12 20:38:31 |

**Obrázek 5.5. Tabulka LOG s příslušnými daty**

## 5.2. Technika zobrazení

Máme tedy připravena data a jejich formát pro zobrazení. Nyní je třeba zvolit vhodnou metodu pro samotné vykreslení stromu. Takových řešení je značné množství, proto se zde pokusím diskutovat jen vybrané možnosti.

### 5.2.1. C++ aplikace

Velmi elegantní řešení, co se týče rychlosti - algoritmus ve zkompilované aplikaci bude bezpochyby velmi rychlý. Netroufám si posuzovat jiné aspekty C++ související s problémem vizualizace, nicméně vytvoření takové plně interaktivní aplikace pro zkušeného programátora není náročné. Bohužel problémem jsou dvě zásadní věci: přenositelnost aplikace mezi operačními systémy a její dostupnost. Aplikace by se zřejmě musela vytvořit pro různé prostředí zvlášť. Uživatel, který by ji chtěl využívat, by si musel nejprve stáhnout správnou verzi a nainstalovat. Vhodnější bude jazyk založený na webových technologiích.

### 5.2.2. Java applet

Java je objektově orientovaný jazyk vyvinutý firmou Sun Microsystems a byla navržena jako snadno přenositelný jazyk. Vytvoření java appletu si opět netroufám příliš do hloubky posuzovat. Výhodou je vyšší dostupnost, tedy pokud bychom vytvořili applet, který bude dostupný na webových stránkách, bude stejný pro všechny OS. Výhodou je, že se jedná o plnohodnotnou aplikaci, tzn. může otevírat jakékoli síťové spojení na jakémkoli portu a číst data. Také samotné vykreslování není nezdolatelnou záležitostí a je možné vytvořit uživatelsky přívětivé rozhraní. Problémem je, že uživatel musí mít nainstalovaný virtuální java stroj (JVM - Java Virtual Machine), tedy Java Runtime Enviroment.

### 5.2.3. Flash

Flash je vektorový grafický formát, určený zeména pro tvorbu animací a potažmo i webových stránek. Jeho nespornou výhodou je i podpora ActionScript, což je jazyk na řízení animací podobný JavaScriptu. Je možné tedy tvořit pokročilé interaktivní animace. Jeho další předností je podpora v dnešních prohlížečích, kdy je sice potřeba plug-in (zásuvný modul), ale mnohdy je plug-in součástí přímo instalace prohlížeče. Tato technologie se jeví jako vhodná pro potřeby vizualizace dat, nicméně má jednu zásadní nevýhodu - formát Flash aplikace (SWF) je uzavřený, při vývoji flash animace (interaktivní aplikace) je možné používat jen aplikace, které umožní export SWF.

V dnešní době se umí Flash "spojit" s databází a číst z ní data. Tato řešení jsou jednak na bázi již zmiňovaného ActionScriptu (například ActionSrcipt knihovan Matta MacLeana asSQL[1]) a jednak ve spolupráci s produkty třetích stran (například AMFPHP[2]). Řešení ale není dle mého názoru ideální, je jisté, že skriptovací jazyk na bázi Javascriptu nebude nabízet dostatečné množství nástrojů pro zpracování dat a využití knihoven třetích stran není pro tuto práci příliš šťastné.

### 5.2.4. PHP skript a GD knihovna

PHP je skriptovací jazyk určený pro tvorbu webu (stránek a webových aplikací), který pracuje na straně serveru. Mezi výhody PHP patří jeho relativní jednoduchost, podpora velkého množství webových a souvisejících standardů a technologií, otevřenost a také dobrá komunikace s databázemi. Je tedy vhodný pro účely vizualizace, pokud bychom dynamicky generovali webové stránky, které by data zobrazovaly. Mezi nevýhody patří zejména jeho rychlost, jelikož se jedná o interpretovaný, nikoli kompilovaný jazyk.

[1] http://code.google.com/p/assql/
[2] http://www.amfphp.org/

GD knihovna je, dalo by se říci, "nadstavba" PHP pro práci s grafikou. Umožňuje vytváření základních grafických obrazců a manipulaci s nimi, jakož i práci s externími soubory (vložené obrázky GIF, JPEG apod.). Její nevýhodou je rychlost a náročnost na server a zejména to, že výstupem GD knihovny je většinou rastrový obrázek (opět JPEG GIF, PNG apod), takže se příliš nedá mluvit o interaktivitě takové grafiky. Je vhodná spíše pro dynamické vykreslování grafů, úpravy vlastností obrázků před vlastním vykreslení na webové stránce apod.

### 5.2.5. PHP a generování XHTML (SVG)

Velmi zajímavou možností je PHP sloužící k vygenerování XHTML. XHTML (Extensible Hypertext Markup Language – „rozšiřitelný hypertextový značkovací jazyk“) je značkovací jazyk pro tvorbu hypertextových dokumentů v prostředí WWW vyvinutý konsorciem W3C. XHTML je jakousi nadstavbou respektive odvozením značkovacího jazyka XML, čímž umožňuje vkládání i jiných, na XML založených formátů v jednom jediném dokumentu. Jeho nevýhodou je velmi striktní stavba a ne jasně definovaná podpora u nejrozšířenějšího webového prohlížeče Microsoft Internet Exploreru (bude diskutováno dále).

Princip PHP je podobný jako u všech skriptovacích jazyků, výstupem programu je většinou textová či grafická informace, v tomto případě většinou webová stránka. V PHP není nutné definovat datové typy proměnných, čímž výrazně usnadňuje práci.

SVG (Scalable Vector Graphics) je otevřený vektorový grafický formát založený na syntaxi XML. Zdrojový kód grafiky je tedy psán v textovém režimu do tagů podobně jako HTML a teprve v samotném prohlížeči je jazyk interpretován na grafiku a ta převedena z vektorů na klasický rastrový formát. Jeho nespornou výhodou je podpora Javascriptu a animací, čímž umožňuje vzniknout kvalitní interaktivní grafice.

Díky společnému základu XHTML a SVG (tedy XML) a variabilitě XHTML je možno tyto dva formáty kombinovat. Pokud tedy budeme v PHP dynamicky, na základě dat z databáze, generovat webovou stránku založenou na XML standardech, vznikne nám spolu s Javascriptem (skriptovací jazyk na straně webového klienta), interaktivní web s potřebnou vizualizací dat. Tento způsob by mohl být pro tvorbu vizualizace vhodným.

### 5.2.6. Java servlety, JSP a JSTL

## Servlety

Servlety jsou server-side programy napsané v jazyce Java, které se používají ke generování dynamických webových stránek, obdobným způsobem jako je tomu například u PHP. Z toho plyne i podobný způsob použití - na server přijde určitý požadavek na stránku, server zavolá servlet, který vygeneruje příslušný dynamický obsah stránky a pošle zpět klientovi.

Oproti již zmiňovaným appletům, které běží na straně klienta v Java Virtual Machine a jejichž nahrávání je pomalé a technologie je spíše na ústupu, servlety a obdobné technologie postavené na jazyku Java se stále více rozmáhají. Servlety mohou například přečíst a zpracovávat data z odeslaných klientových webových formulářů, zajistit komunikaci s databází, formátovat výsledky například do HTML výstupu, pracovat s cookies a podobně. Jejich výhodou je, že jsou platformě nezávislé, drobné odlišnosti jsou pouze v jednotlivých verzích (2.1, 2.2) [15].

V porovnání s CGI skripty (PHP apod.), kdy se každý požadavek vyřizuje v novém procesu, což může být poměrně náročné na výkon operačního systému a stroje, na kterém běží, u servletu běží pouze virtuální stroj Java a jednotlivé požadavky obsluhuje vlákny. Výhodou je také, že data se "neztratí" potom, co byla v pořádku odeslána klientovi, jako je tomu u PHP, data zůstávají v paměti i po dokončení transakce a je možné je využít i později. Servlety jsou přenositelné a platformě nezávislé. Zkompilovaný program není instrukcemi pro daný procesor, ale instrukcemi Java Virtual Machine, teprve až po zavolání je program přeložen do binárních instrukcí daného procesoru [15]. Java Server Pages (JSP),

## JSP

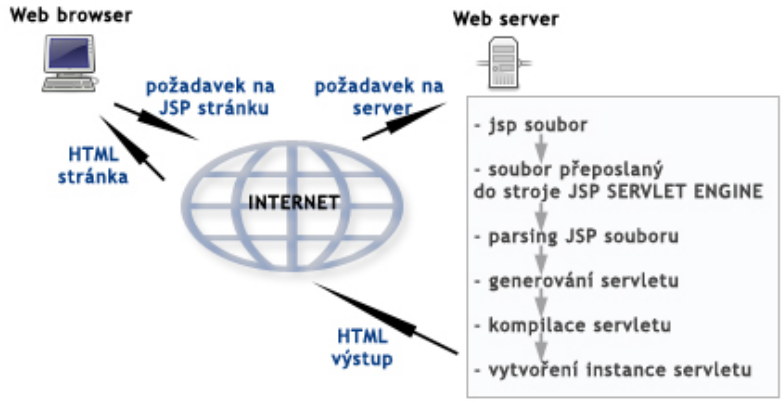
Java Server Pages je nástroj pro směšování dynamického obsahu napsaným v Javě se statickým HTML obsahem. V literatuře [15] je uvedeno, že slouží zejména k tomu, pokud chceme do v podstatě dynamického obsahu vkládat jen dynamické fragmenty a je zbytečné použití servletů. JSP umožňují ovšem vytvářet velice složité a sofistikované webové projekty. Server Pages byly navrženy tak, aby byl pokud možno oddělen vlastní logický kód od HTML

obsahu. Pro vkládání je tedy použito speciálních tagů, které reprezentují dynamický obsah. Filozofie, že bude oddělena prezentační vrstva od vrstvy logické a datové, tedy že budou webové stránky vytvářet HTML kodéři, kdežto o programovou logiku se postará programátor, je bezpochyby dobrá, nicméně domnívám se, že technologie není tak daleko, aby se obsah striktně oddělil, tudíž HTML kodér musí mít stále znalosti o Javě, což je poměrně nevýhoda.

Princip funkce spočívá v tom, že při zavolání JSP souboru se vytvoří servlet, který se zkompiluje a spustí. Toto se provede pouze při prvním volání, při dalším volání se použije již zkompilovaná verze. První spuštění tedy většinou provede programátor/kodér, aby se urychlilo další spouštění.

Výhodou oproti CGI skriptům je, že Java je robustnější a je již předpřipravena celá řada tříd například pro práci s databází, SMTP a podobně. Jako výhodu lze považovat i vestavěný aparát pro rozeznání, zda-li se jedná o parametry skriptu s využitím metody GET nebo POST. Narozdíl od konkurenční technologie ASP od firmy Microsoft je technologie opět univerzální a přenositelná, není tedy vázána na jeden typ serveru (IIS) a operační systém.

Oproti servletům je JSP technologie jednodušší a přehlednější, nicméně díky tomu, že JSP je stejně přeložena na servlet, jsou možnosti v podstatě rovnocenné.




**Obrázek 5.6. Schéma principu činnosti JSP**

## JSTL

JSP podporuje tvorbu vlastních uživatelských značek. Tyto uživatelské značky jsou založeny na formátu XML a slouží k používání opětovnému volání knihoven. Takto je zajištěno ještě vyššího oddělení části prezentační od logické vrstvy. Tagy zapouzdřují části kódu, které se často vyskytují a tím umožňují větší komfort při tvorbě webové aplikace. Případné změny nebo úpravy se totiž provedou na jednom místě a projeví se na všech místech, kde je kód pomocí značky použit.

JSPTL/JSTL JSPTL (standard JavaServer Pages Tag Library) je knihovna, která zapouzdřuje většinu základních funkcí JSP stránky, jako jsou cykly, podmínky, výstupy, proměnné a práci s XML. Umožňuje i vytváření vlastních tagů dle programátorových potřeb. Jedná se tedy o velmi silný nástroj pro přehlednou tvorbu zejména internetových aplikací. Pro představu následuje příklad cyklu se zapsáním proměnné, podmínkou a vložením další JSP stránky.

```
<c:forEach var="node" items="${node.childs}" varStatus="loop">
 <c:set var="node" value="${node}" scope="request"></c:set>
 <c:if test="${node.name == parametr}">
  <c:set var="node_start" value="${node}" scope="request"></c:set>
 </c:if>
 <jsp:include page="node_loop.jsp" />
</c:forEach>
```

Zmíněné technologie založené na Javě jsou samozřejmě mnohem složitější resp. jejich podrobnější popis a vysvětlení se nevejde do rozsahu této práce. Mohly by zde být popsány podrobněji metody používané v servletech a princip jejich činnosti, možnosti, implicitní objekty, direktivy a značky JSP, práce s cookies a sessions a další

aspekty, nicméně i na základě zde uvedeného nastínění této problematiky je jasné, že pro vizualizaci bude vhodné použít technologií založených na jazyku Java, a to zejména ze dvou důvodů.

Prvním z nich je, že se jedná o moderní technologie s dostatečnou robustností, aby splňovala všechny předpoklady na vizualizaci. Jsou to technologie navrženy pro tvorbu webových aplikací, výhodou je také přenositelnost, multiplatformnost. Pokud tedy použijeme JSP/JSTL ve spolupráci s XHTML a grafickým formátem SVG, je technologie vhodná pro vizualizaci. Je nutné brát také zřetel na přehlednost kvůli pozdějším úpravám.

Druhým a také podstatným důvodem je, že ostatní části projektu IPTV (sběr dat, práce na hierarchické sumarizaci) jsou řešeny za pomoci těchto technologií. Je tedy vhodné, aby byly použity i zde.

# 6. Formáty pro vlastní výstup vizualizace (XHTML a SVG)

V předchozích kapitolách bylo popsány důvody a principy zvolených metod, zejména způsob ukládání dat pro vizualizaci a forma jejich zpracování k přípravě zobrazení. V této kapitole budou podrobněji popsány technologie použité při konkrétním použití pro vlastní vizualizaci se zaměřením na konkrétní popis důležitých částí související s řešeným problémem.

## 6.1. XHTML

Jak již bylo řečeno v předchozí kapitole, XHTML a HTML se v některých aspektech liší. Není účelem této práce se do hloubky zaobírat rozdíly, nicméně je nutné osvětlit některé odlišnosti a specifika, která jsou pro zvolený způsob vizualizace zásadní.

- DTD (Document Type Definitions) – pravidla pro psaní XHTML kódu jsou v podstatě stejná, jako pro notoricky známé HTML. S tím rozdílem, že tagy (např. `<head></head>`, `<iframe/>`) musí být vždy uzařeny a uváděny malými písmeny. Taktéž je zakázáno používání některých atributů (`width=""`, `border=""`, apod.), jelikož XHTML je úzce svázáno s používáním kaskádových stylů CSS (Cascading Style Sheets). Pro zobrazení stránky bude použito striktního „režimu“, dokument se bude kontrolovat proti „nejradikálnějšímu“, avšak čistému a modernímu DTD. Předpokladem pro tento výběr je, že uživatelé webu mají novější verze internetového prohlížeče (MSIE 6 a vyšší, Firefox, Opera apod.). Typ dokumentu v záhlaví tedy vypadá takto:

  ```
  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
  ```

- O tom, jakým způsobem je dokument zobrazen, tedy o jaký soubor se vlastně jedná, informuje tzv. MIME typ, který se posílá v hlavičce HTTP protokolu. Ty obvykle odešle server, který má definováno, jaké přípony odpovídají jakým MIME typům. MIME XHTML souboru je

  ```
  application/xhtml+xml
  ```

  Navíc díky tomu, že XHTML vychází z XML, je nutné v každém dokumentu uvést prolog:

  ```
  <?xml version="_verze_" encoding="_kódování_" standalone="_def_styl?_"?> <!-- obecně -->
  <?xml version="1.0" encoding="utf-8" standalone="yes" ?> <!-- pro účel této práce -->
  ```

  Zde ovšem vzniká poměrně zásadní problém. Nejpoužívanější prohlížeč Microsoft Internet Explorer (dále již IE nebo MSIE) si neumí poradit s takovým MIME typem a nabídne uživateli uložení souboru místo toho, aby jej korektně zobrazil. To je nežádoucí. Řešením tedy je „nepodsouvat“ prohlížeči, že se jedná o XML soubor a vynechat v záhlaví dokumentu prolog [31]. Specifikace XHTML umožňuje vynechání prologu pouze, pokud se jedná o kódování utf-8, což je pro naše účely postačující. Poté bude již dokument zobrazen korektně (odzkoušeno na IE6.5 a IE7). Na to bude přihlíženo při JSP generování webové stránky s vizualizací (viz dále).

## 6.2. JavaScript

JavaScript je programovací jazyk, jenž slouží k dynamické změně internetových stránek na klientově straně, tzn. program se odešle ve stránce klientovi a teprve poté je vykonáván. O jeho interpretaci se tedy stará prohlížeč. Z toho můžou vzniknout problémy, jelikož opět existují odlišnosti u typů prohlížečů.

Kód JavaScriptu se nepřekládá, je tedy viditelný a volně přístupný v rámci zdrojového kódu stránky. To může být výhodné zejména v tom, že je takto k dispozici velmi velké množství již předem připravených řešení, na druhou stranu je ovšem těžké skrýt vlastní způsoby řešení a provádět autorizaci návštěvníků stránek a implementovat různé procedury pro bezpečnost.

Javascript umožňuje přistupovat k jednotlivým částem (X)HTML či SVG dokumentu (DOM - Document Object Model) a upravovat je nebo měnit, a to jednak po spuštění stránky nebo jako reakce na určité události (kliknutí myší, přejetí myší přes objekt apod.).

Document Object Model - objektový model dokumentu je aplikační programové rozhraní pro práci s jakýmkoli SGML (HTML, XHTML, XML, SVG ...) dokumentem. To znamená, že se jedná o hierarchii objektů (elementy, atributy a jejich hodnoty), která umožňuje prohlížeči pracovat s jejich vlastnostmi a metodami. DOM umožňuje přistupovat k jednotlivým objektům XML (XHTML) dokumentu a pracovat s nimi. Díky jeho přísně stromové struktuře je možné přistupovat k přesně určeným uzlům (objektům) - nadřazeným, podřazeným a rovnocenným elementům.

Nevýhodou ovšem je již zmíněný problém s kompatibilitou mezi prohlížeči. To, co funguje v jednom, má problémy v jiném, navíc uživatel si může JavaScript úplně vypnout. Skripty také nemohou využít síťových spojení, pouze přesměrování na jinou stránku, nemohou také na disku upravovat a vytvářet soubory a podobně [33].

I přes zmíněné nevýhody je využití JavaScriptu pro účel této práce dostačující jako doplňková funkce pro manipulaci s objekty vizualizace. V tomto projektu bude použit na interaktivitu obrázku SVG. Například při přejetí myší nad objektem bude aktivována funkce, která zobrazí informace o jednotlivém uzlu (viz dále).

## 6.3. SVG

Formát SVG je poměrně mladý a neustále se vyvíjí, jeho první specifikace vyšla v roce 2002. Scalable Vector Graphics je platformou pro vektorovou dvoudimenzionální grafiku, která je složena ze dvou částí: formát založený na XML (extensible Markup Language) a část pro programování API pro aplikace, které tento formát umí podporovat. SVG je vektorová grafika, což znamená, že formát neobsahuje přímé informace o každém jednotlivém pixelu nebo úseku obrázku, ale obsahuje pouze údaje o vektorech, tedy křivkách v souřadnicovém systému a jejich výplních. Teprve v samotném prohlížeči je grafika převedena z vektorů na klasický rastrový formát. Bezesporu největší výhodou je, že obrazovou informaci není třeba zapisovat složitým způsobem, jak je tomu u rastrové grafiky - stačí popsat vektor příslušnými souřadnicemi a údaje o jeho výplni. Proto byl pro zápis samotné grafiky zvolen formát XML, který je určený pro univerzální výměnu informací, přičemž zároveň zaručuje maximální otevřenost a přizpůsobení se novým potřebám. SVG lze tedy vytvářet v obyčejném textovém režimu prostým zápisem těchto elementů pod sebe tak, jak mají být zobrazeny. Aby se zaručila maximální univerzálnost interpretace grafiky, elementy již nemají libovolné názvy jak je tomu u XML, ale jsou vytvářeny podle přesně daných pravidel. Díky tomu vznikl otevřený a strukturovaný formát, který svou flexibilitou a přehledností vyhovuje všem oblastem elektronického grafického vyjádření.

Na první pohled by se mohlo zdát, že SVG je sice výborný formát pro základní grafické vyjádření, ale nenabízí příliš velké možnosti ve formátování a automatických upravách (filtrech) jako klasická bitmapová grafika. Díky přizpůsobivosti však SVG tento nedostatek rychle dohání. Navíc, jak již bylo zmíněno, SVG pracuje s vektory a ne s jednotlivými pixely, což se projeví zejména na kvalitě, pokud je obrázek trasformován nebo zvětšován. Bitmapové obrázky přitom většinou ztrácejí na kvalitě a přesnosti zobrazení, SVG téměř nikoliv. Velkou výhodou je strukturovatelnost SVG. Grafik je na základě přehledné struktury SVG schopen jednotlivé objekty vzájemně seskupovat, členit a dokonce jednoduše dodatečně upravovat. SVG pracuje v souřadnicovém systému, takže je možné provádět dodatečné změny pouze přepisem souřadnic jednotlivých objektů. Díky tomu, že zdrojovým kódem pro zobrazení SVG obrázku je prostý text, lze jednotlivé objekty (které mají jednoznačný identifikátor "`id`") vyhledávat. Textový formát se vyznačuje minimálními nároky na velikost paměťového prostoru, což je příčinou tak prudkého rozvoje u mobilních telefonů a webových technologií, kde je nutnost úspory přenášených dat. Navíc SVG nabízí i komprimovanou verzi (soubor s příponou ".svgz") založenou na algoritmu GZIP, která ještě více zmenšuje datový objem. Zajímavou vlastností je i interaktivita grafiky, kde se SVG stále vyvíjí. Například kliknutím na obrázek se provede změna barev apod. Tak je možno, že obrázek pomocí událostí a JavaScriptů určitým způsobem "komunikuje" s tím, kdo si jej prohlíží. SVG úzce spolupracuje s kaskádovými styly CSS, což umožňuje designérům měnit automaticky vzezření celých skupin prvků a měnit jejich vlastnosti podle potřeby umístění. Navíc nabízí možnost animací, čímž konkuruje dnes hojně používaném formátu Flash. Nespornou výhodou je také automatické generování jiných formátů ze SVG a také podpora velkých firem (Adobe, Corel, Hewlett - Packard, Bitflash, Google atd.), což do jisté míry zaručuje kompatibilitu a rozvoj SVG. Na druhou stranu, SVG je technologie poměrně nová, takže by se jí dala vytknout určitá nedotaženost a slabá podpora v některých aplikacích, zejména prohlížečích a grafických editorech, která brání jejímu masovému nástupu.

### 6.3.1. Podpora SVG v prohlížečích

Podpora zobrazení SVG souborů je v současných prohlížečích realizována dvěma způsoby: přídavným zásuvným modulem či implementací SVG přímo do prohlížeče. Součástí této práce není podrobný výklad, kde a za jakých podmínek SVG funguje, proto zde bude uveden jen základní přehled.

# 6.3.1.1. Plug - in podpora

Plug-in podpora vychází z již známé technologie, kdy konkrétně v tomto případě prohlížeč zajišťuje rozhraní vstupu a výstupu, kdežto samotné zpracování je zajišťováno samotným zásuvným modulem. Mezi nejznámější a nejpoužívanější patří Adobe SVG Viewer[1] (v současnosti ve verzi 3.03). Tento plug-in zahrnuje kompletní specifikaci SVG, avšak není k dispozici pro jiné platformy, než některé verze Windows a MacOS. V HTML vyžaduje nestandardní vložení pomocí tagu `<embed>` a navíc není podporován všemi prohlížeči. Vkládání obrázku a podpora automatického vyžádání vložení Plug-inu do webového prohlížeče se tedy provádí tagem `<embed name="" pluginspage=http://www.adobe.com/svg/viewer/install/ src="" type="image/svg+xml">`. Nutno podotknout, že subjektivně se odezva naší aplikace jeví rychlejší při použití ASV, než pokud je grafika vykreslena nativně přímo prohlížečem (viz dále)

## 6.3.1.1.1. Internet Explorer a OS od firmy Microsoft

Dalo by se říci, že v současné době je firma Microsoft na špici co se týče možností SVG ve svém prohlížeči, i přes to, že standardně se svým prohlížečem nedodává plug-in (zásuvný modul) ani se nezabývá přímou implementací SVG. Mnou testovaná verze Internet Explorer 6.5 a 7 si s dodatečně nainstalovaným Adobe SVG Viewerem skvěle rozumí a díky tomu v současnosti nabízí podporu téměř všech částí SVG včetně animací.

Podobná situace je s podporou SVG v operačním systému jako takovém. Zatímco nejnovější verze grafického rozhraní KDE pro Linux dokáží přímo zobrazit SVG soubory (včetně náhledů), Microsoft zaostává. Ve verzích operačního systému Windows XP takováto implementace není, ve Windows Vista rovněž ne.

## 6.3.1.1.2. Mozilla a Mozilla Firefox (starší jádra)

Vývojáři těchto prohlížečů pracují na nativní podpoře SVG, která ale není zdaleka dokončená, proto je možno nativní podporu obejít, respektive přidat možnost zobrazování SVG i do verzí, které ještě nativní podporu nemají ( například Firefox verze 1.0.7 a nižší ). Řešením je opět instalace Adobe SVG Vieweru.

Jak již bylo řečeno, pomocný plug-in od Adobe se do Internet Exploreru nainstaluje automaticky, do Mozilly však nikoliv. Proto je nutné do složky "<instalaceMozilly>/plugins" zkopírovat soubor "NPSVG6.dll", který se po instalaci Adobe Vieweru nachází v adresáři "C:/Program Files/Common Files/Adobe/SVG Viewer 6.0". Pro současné verze SVG vieweru ( 3.0 - 3.3 ) může být internetový prohlížeč nestabilní, nicméně s připravovanou verzí 6.0 by se měla stabilita a spolupráce podstatně vylepšit.

# 6.3.1.2. Nativní podpora

Druhým způsob je tzv. nativní podpora, což znamená, že SVG podpora je implementována přímo do prohlížeče. To se jeví jako lepší způsob, jelikož uživatel nemusí dodatečně instalovat zásuvný modul. Avšak žádný z výrobců dnes nejvíce používaných prohlížečů, které jsem testoval, nebyl schopen implementovat celou specifikaci SVG.

## 6.3.1.2.1. Mozilla Firefox

Nativní podpora existuje od verze 1.5. Se zabudovanou knihovnou Cairo, která umožňuje solidní podporu SVG, na které se navíc velmi rychle pracuje. Opět sice není implementována zdaleka celá specifikace SVG 1.1[2] (filtry, animace atd. - mozillou podporované elementy a atributy SVG jsou na webových stránkách projektu[3]). Výhodou je také podpora ECMA E4X, což po zavedení do SVG podstatně zvýší komfort ovládání událostí pomocí skriptů. Bohužel práce na podpoře SVG u Firefoxu nezaznamenávají příliš výrazné změny.

## 6.3.1.2.2. Opera

Starší verze Opery (např verze 7) podporovaly částečně Adobe plug-in. V nových verzích prohlížeče (9 a dále) je zavedena podpora specifikace SVG 1.1. Podle mého názoru se jedná o nejlepší řešení, co se týče implementace SVG. Navíc tento prohlížeč je k dispozici i pro jiné platformy, např. Linux a Mac OS.

[1] http://www.adobe.com/svg/viewer/install/
[2] http://www.w3.org/Graphics/SVG/
[3] http://developer.mozilla.org/en/docs/SVG_in_Firefox_1.5

Opera se také stává stále významnější se svou verzí prohlížeče pro mobilní zařízení, kde se nabízí široké pole využití SVG. Instaluje se například v některých typech mobilních telefonů Nokia s operačním systémem Symbian.

# 6.3.1.3. Identifikace SVG prohlížeče

V předchozích podkapitolách byly zmiňovány vybrané prohlížeče a jejich vlastnosti, co se týče podpory grafického formátu SVG. Nejednotnosti však brání masovému zavedení SVG na web, jednak v důsledku zmiňovaných rozdílů a nutnosti přizpůsobit a otestovat grafiku pro každý jednotlivý prohlížeč a jednak kvůli tomu, že ne každý běžný "uživatel" internetu bude vůbec používat prohlížeč podporující tento grafický formát.

Pro zobrazení tedy budou použity jen ty prvky SVG, které je schopna vykreslit Opera i Firefox. Uživatelé MSIE jsou vyzváni k instalaci zásuvného modulu (ovládacího prvku ActiveX od Adobe).

## 6.3.2. Vkládání SVG do struktury HTML

V předchozí podkapitole bylo rozebíráno zobrazení SVG formátu jako takového, nicméně dalším problémem je vložení SVG grafiky do HTML stránek tak, aby bylo zaručeno správné zobrazení grafiky ve všech prohlížečích. Existují opět dva způsoby vložení SVG, a to buď zdrojovým kódem SVG přímo do kódu HTML nebo vložení externího obrázku tak, jak je zvykem u jiných grafických formátů. Nicméně i druhý způsob má svá specifika. Rozdílné jsou také požadavky jednotlivých prohlížečů.

# 6.3.2.1. Přímé vložení kódu

SVG lze dopsat přímo do HTML (XHTML) kódu webové stránky za použití jmenného prostoru SVG: `xmlns:svg="http://www.w3.org/2000/svg"`. Nevýhodou je ale toto použití jmenného prostoru. To totiž znamená, že je nutno do každého SVG tagu přidat kód "`svg:`", např místo `<rect...></rect>` se použije `<svg:rect....></svg:rect>`. Navíc se musí lišit i celková struktura HTML stránky pro různé prohlížeče a i při správném vložení je SVG omezeno v možnostech skriptování a animací.

Pro náš účel by bylo takové vložení SVG možné, jelikož nebudeme vykreslovat příliš složité obrazce.

## 6.3.2.1.1. Internet Explorer s Adobe SVG Viewer

Internet Explorer sám o sobě neumí zpracovat SVG grafiku a přímo vloženou do HTML nepozná ani Adobe SVG Viewer, proto je nutné jej "svázat" se jmenným prostorem SVG, a to pomocí zápisu umístěného nejlépe někde na počátek dokumentu, např do `<head>`:

```
<object id="AdobeSVG" classid="clsid:78156a80-c6a1-4bbf-8e6a-3cd390eeb4e2"></object>
<?import namespace="svg" implementation="#AdobeSVG"?>
```

Při testování aplikace vizualizace v různých prohlížečích bylo ale zjištěno, že i přes to, že je SVG vloženo do struktury HTML dle normy, není možné používat JavaScript pro řízení aplikace u prohlížeče Microsoft Internet Explorer. Adobe SVG Viewer je sice svázán se jmenným prostorem SVG a grafiku vykresluje správně, skripty ignoruje. Proto bylo přistoupeno k jiné možnosti vložení grafiky, které s sebou ale přináší některé nevýhody.

# 6.3.2.2. Vložení externího SVG souboru

Vložení externího souboru do HTML je méně komplikované, než předchozí řešení a navíc více univerzální. Většina prohlížečů podporuje HTML tag "`object`", např:

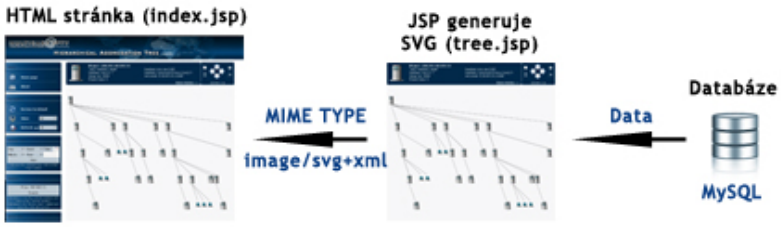
```
<object data="tree.svg" type="image/svg+xml" width="1000" height="1000" />
```

Pro náš příklad, kdy je nutné generovat SVG dynamicky v určitých časových intervalech není vhodné vygenerovanou grafiku ukládat na disk a poté zobrazovat. Vhodnější je využít JSP stránku, která se bude "vydávat" za SVG a bude data generovat dynamicky, aniž by bylo nutné mezistupně uložení na disk.

```
<c:url value="svg_obj/tree.jsp" var="treeURL">
 <c:param name="ip" value="${node.info.ip}" />
 <c:param name="gip" value="${node.info.ip}" />
```

```
 <c:param name="view_type" value="${URLHandler.view_type}" />
 <c:param name="rate" value="${URLHandler.rate}" />
 <c:param name="mi" value="${URLHandler.mi}" />
 <c:param name="hh" value="${URLHandler.hh}" />
 <c:param name="dd" value="${URLHandler.dd}" />
 <c:param name="mm" value="${URLHandler.mm}" />
 <c:param name="yy" value="${URLHandler.yy}" />
</c:url>
<c:url value="index.jsp" var="dateURL">
 <c:param name="mi" value="${URLHandler.mi}" />
 <c:param name="hh" value="${URLHandler.hh}" />
 <c:param name="dd" value="${URLHandler.dd}" />
 <c:param name="mm" value="${URLHandler.mm}" />
 <c:param name="yy" value="${URLHandler.yy}" />
</c:url>

<!--[if IE]>
 <embed id="svg1" src="${treeURL}" pluginspage="http://www.adobe.com/svg/viewer/install/"
 type="image/svg+xml" width="1000" height="1000"> </embed>
<![endif]-->
<!--[if !IE]> -->
 <div class="svg" id="svg1" name="svg1"><object data="${treeURL}"
 type="image/svg+xml" width="1000" height="1000"></object> <!-- <![endif]-->
```

Předchozí úryvek zdrojového kódu HTML stránky ukazuje vložení SVG přes JSP dynamickou stránku `tree.jsp`. V příkladu lze vidět nevýhodu tohoto způsobu vložení - jednak by měly být pevně zadány rozměry `width` a `height` vložené grafiky a navíc veškeré parametry pro vizualizaci musí předávat JSP stránce, která je metodou GET převezme. Případné změny či přidání parametrů se musí tedy v kódu upravit jednak zde při vložení SVG a jednak na vstupu samotného souboru `tree.jsp`. Následující obrázek naznačuje princip vložení SVG.




**Obrázek 6.1. Princip vložení SVG grafiky**

## 6.4. Dynamika vykreslení zvolenou metodou

Jako jeden z požadavků na zobrazení bylo, aby vizualizace byla dynamická, tzn. aby se zobrazení měnilo s měnící se sítí uzlů (měnícími se daty o síti). XHTML takovou možnost bohužel primárně nepodporuje, bude tedy nutné stránku vždy obnovit, aby se její obsah načetl znovu a došlo k aktualizaci.

Implicitně se stránka obnovuje zmáčknutím tlačítka "obnovit" v prohlížeči nebo stisknutím tlačítka F5. Uživatel by musel pro každé nové zobrazení použít zmiňované obnovení, což není ideální. Dalším možným způsobem obnovy stránky je pomocí přesměrování v Javascriptu.

```
<script>
 function obnoveni() {
  window.location.href="http://www.url.cz/cokoliv";  //presmerovani na adresu
 }
 setTimeout(obnoveni(),5000);    //nastaveni casovace na 5s
</script>
```

Po nahrání stránky se spustí časovač a po uplynutí určené doby se funkcí `obnoveni()` stránka přesměruje (obnoví). Problém ovšem vzniká, pokud má uživatle Javascript zakázaný.

Daleko elegantnějším řešením je použití meta tagu na přesměrování. Meta tagy slouží k definování základních vlastností stránky (kódování, jazyk, klíčová slova apod.). Tag na přesměrování je:

```
<meta http-equiv="refresh" content="4;url=http://www.url.cz/cokoliv"> //obecne presm. za 4s
<meta http-equiv="refresh" content="10"/> //presmerovani sam na sebe za 10s
```

První parametr atributu "content" je čas v sekundách udávající, za jakou dobu se má přesměrovat a druhý vyjadřuje, kam se má přesměrovat. Pokud je druhý parametr prázdný, přesměruje se na stávající stránku, čehož bude využito i pro naše účely.

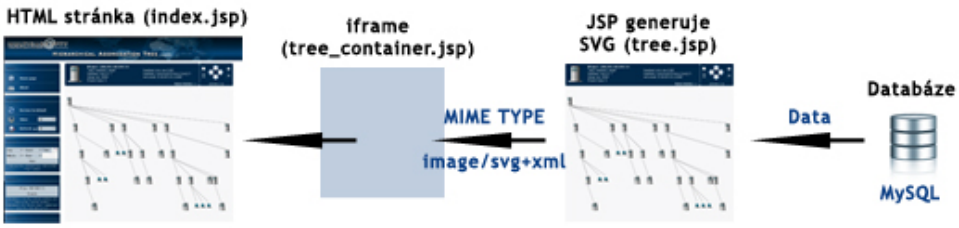
Takto tedy bude dosaženo toho, že se stránka obnoví v pevně daném okamžiku a vykreslování sítě tak bude "pseudodynamické". Pokud nastavíme dostatečně malý časový okamžik, stránka se bude v porovnáním se získáváním dat o síti (PING) obnovovat zanedbatelnou dobu a dosáhneme potřebné dynamiky.

Problém v obnovení je ovšem ten, že stránka se obnovuje celá. Obnova celé webové stránky, řekněme každých 10s, by působila rušivě. Toto lze obejít pomocí vnořeného plovoucího rámu. Tag <iframe> bohužel specifikace XHTML zakazuje, proto je třeba toto obejít a externí stránku vložit pomocí tagu <object>.

```
<!--[if IE]>
 <object classid="clsid:25336920-03F9-11CF-8FD0-00AA00686F13" data="tree_container.jsp"
 width="1000" height="1000">
  <p>backup content</p>
 </object>
<![endif]-->

<!--[if !IE]> <-->
 <object type="text/xhtml" data="tree_container.jsp" width="1000" height="1000">
  <p>backup content</p>
 </object>
<!--> <![endif]-->
```

Zde je navíc využito podmíněných komentářů, což je nástroj speciálně využitelný pouze v MSIE (verze 5 a vyšší), který podmíněné komentáře umí používat. Takto tedy byla do stávající stránky vložena stránka zobrazeni_xhtml.php, která obsahuje SVG a bude se v pravidelném časovém okamžiku obnovovat.

Od tohoto způsobu obcházení obnovy celé stránky bylo nakonec v projektu upuštěno. Důvodem jsou v předchozí kapitole zmiňované parametry předávané stránce tree.jsp potřebné k vizualizaci. Tyto parametry by se tedy musely zpracovávat a předávat celkem třikrát. Nejprve ve vlastní stránce index.jsp, poté by byly předány stránce tree_container.jsp a poté samotné vizualizaci SVG, stránce tree.jsp. Toto řešení je z hlediska přehlednosti aplikace nepřípustné.




**Obrázek 6.2. Schéma možného vložení přes <iframe>**

## 6.5. Vykreslení dat

V algoritmu tedy budeme za sebou po řadě vypisovat nejprve hlavní uzel a poté jeho potomky. Pro každého potomka opět procházíme strukturou a hledáme jeho potomky a ty také po řadě vypisujeme. Využíváme rekurzivního volání funkce pro vykreslení a strom je v podstatě vykreslován metodou PRE-ORDER.

Problémem je, že v JSP v podstatě nejsou přímo definovány funkce, nemůžeme tedy funkci volat jednoduše rekurzivně, jak je obvyklé v programovacích jazycích. Toto lze ovšem obejít opětovným vkládáním fragmentu pro vykreslení ve smyčce do sama sebe, jak je naznačeno v následujícím příkladu:

```
<c:forEach var="node" items="${node.childs}" varStatus="status" >
 <c:set var="node" value="${node}" scope="request"/>
```

```
...

 <jsp:include page="node.jsp"/>
</c:forEach>
```

Využita je smyčka forEach, tedy "pro každého potomka uzlu (`node`) vlož stejný cyklus forEach".

Výsledný SVG zdrojový kód pro vykreslení uzlu může vypadat takto:

```
<a xlink:href="192.168.2.1&amp;view_type=full&amp;rate=30&amp;mi=10&amp;hh=12&amp;mm=5&amp;
     dd=12&amp;yy=2008" target="_top">
 <g id="node192.168.2.1" transform="translate(0,257)">
  <use xlink:href="#feedback_target" id="x192.168.2.1" onmousemove="pozice_ft(evt,
 '192.168.2.1'), scale_in(evt, '192.168.2.1')" onmouseout="scale_out(evt, '192.168.2.1'),
 zmizet_ft()"/>
  <desc id="type192.168.2.1">Feedback target</desc>
  <desc id="ip192.168.2.1">192.168.2.1</desc>
  <desc id="tree192.168.2.1">1</desc>
  <desc id="gsize192.168.2.1">4000</desc>
  <desc id="res192.168.2.1">90</desc>
  <desc id="shared_tree192.168.2.1">1</desc>
  <desc id="data192.168.2.1">6654</desc>
  <desc id="interval192.168.2.1">5</desc>
  <desc id="activated192.168.2.1">16:44:20 26.2.2008</desc>
 </g>
</a>
```

Zde je dobře vidět čitelnost zdrojového XML kódu, který pro kreslení používá SVG.

Zajímavá je konstrukce elementu `<use>`, ten slouží k odkazování se na již vytvořený definiční objekt. To tedy znamená, že nejprve byl definován objekt s jedinečným `id` "client" někde výše ve struktuře SVG a na něj se poté odkazujeme. Ikona (obrázek) pro klienta byla tedy vytvořena pouze jednou, ale pomocí <use> je opakovaně vykreslován s tím, že jej pouze na souřadnicích posouváme (atribut `transform="translate(0,257)"` ) dle polohy ve stromu.

Vnořené elementy `<desc>` a popřípadě `<title>` se nezobrazují, slouží k dodatečným informacím o elementu, do kterého jsou vnořeny. Toho je využito k zobrazování dat o uzlech. V příkladu lze vidět klientovu IP adresu, polohu ve stromu, zda-li se jedná o klienta nebo o server a další informace. Atributy `id` musí být jedinečné a jsou generovány při vykreslování pomocí JSP.

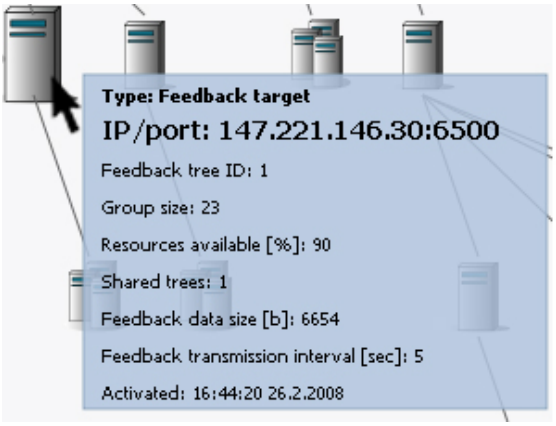
Jedinečnosti `id` je využito v události `onmousemove` (na přejetí myší), kdy se volá Javascript funkce, které je předáno `id` a parametr `evt`. Pokaždé, když se v prohlížeči něco stane, stane se tzv. událost, která je popřípadě předána zájemci o reakci na tuto událost. To, co se stalo, předáváme funkci "pozice_ft", která je v hlavičce XHTML a je definována takto:

```
<script type="text/javascript">
 function pozice_ft(evt, id) {
  var x = evt.clientX;  //souradnice mysi
  var y = evt.clientY;  //souradnice mysi
  x=x+10;
  y=y+10;
  objekt = document.getElementById("extra_text_ft");
  objekt.setAttribute('transform','translate('+x+','+y+')' );
  objekt.setAttribute('visibility','visible' );

  var text1 = document.getElementById('type'+id).firstChild.nodeValue;
  textuzel = document.getElementById("text_type_ft");
  var vystup = document.createTextNode(text1);
  vystup.replaceData(0,0,"Type: ");
  textuzel.replaceChild(vystup,textuzel.firstChild);

... //pro dalsi atributy

  evt.stopPropagation(); //stop evt
  return false;
 }
...
</script>
```

Po zjištění souřadnic myši dojde k zviditelnění poloprůhledného SVG obdélníku s `id="extra_text_ft"` a k zjištění informací z elementů `<desc>` (v příkladu uvedeno zjištění pouze pro jeden element popisu), tedy zjištění informací o uzlu a zkopírování těchto informací do textových elementů k zobrazení:

```
<g id="extra_text_ft" visibility="hidden" font-family="Tahoma" text-
rendering="optimizeLegibility">
 <rect width="250" height="180" fill="lightsteelblue" y="-15" x="-10" opacity="0.85" stroke-
width="1" stroke="#8AA3BC" />
 <text id="text_type_ft" font-size="10" font-weight="bold" y="0" x="0"> </text>
 <text id="text_ip_ft" font-size="14" font-weight="bold" y="20" x="0"> </text>
 <text id="text_tree_ft" font-size="10" y="40" x="0"> </text>
 <text id="text_gsize_ft" font-size="10" y="60" x="0"> </text>
 <text id="text_res_ft" font-size="10" y="80" x="0"> </text>
 <text id="text_shared_tree_ft" font-size="10" y="100" x="0"> </text>
 <text id="text_data_ft" font-size="10" y="120" x="0"> </text>
 <text id="text_interval_ft" font-size="10" y="140" x="0"> </text>
 <text id="text_activated_ft" font-size="10" y="160" x="0"> </text>
</g>
```

Výsledek je poté podobný, jak je tomu na následujícím obrázku:




**Obrázek 6.3. Ukázka výsledného zobrazení s Javascriptovým řízením událostí**

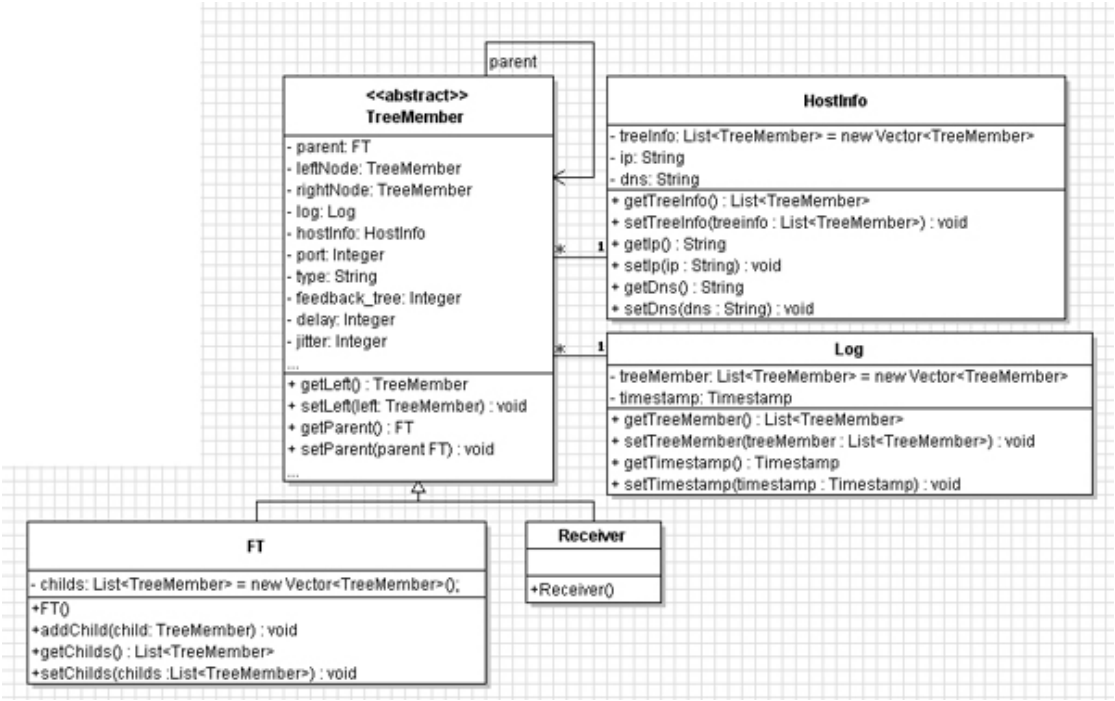
Kompletní okomentovaný zdrojový kód je uveden v příloze této práce.

# 7. Jádro vizualizace (Java, JSP)

V předešlých kapitolách byla navržena a popsána metoda zobrazení, struktura databáze, pro úplnost je třeba popsat, jak funguje samotný algoritmus zobrazení a jak je strom reprezentován ve strukturách programu.

## 7.1. Model stromu

S ohledem na strukturu databáze byla navržena objektová struktura, která reprezentuje vazby ve stromu tak, aby dle takovýchto vazeb bylo možné strom plně rekonstruovat. Vycházejme z toho, že strom má uzly, které mohou být dvou typů: `FT` (Feedback Target, sumarizační uzel) a `Receiver` (přijímač). Tyto typy uzlů se navzájem liší pouze v drobnostech, důležité informace z hlediska jejich polohy ve stromu mají společné. Proto existuje abstraktní objekt `TreeMember` a objekty `FT` a `Receiver` tuto třídu implementují. Struktura je zjednodušeně naznačena na následujícím diagramu tříd.




**Obrázek 7.1. Zjednodušený diagram tříd**

Třída `TreeMember` tedy obsahuje všechny metody pro zápis a získání informací o uzlech. Třída `HostInfo` udržuje a poskytuje metody pro získání "statických" informací o každém uzlu (ip a dns), třída `Log` je určena pro časovou identifikaci uzlu. Abychom plně respektovali navrhnutou databázi, předpokládáme, že na jeden objekt `Log` je navázáno mnoho objektů `TreeMember`. Každý uzel `TreeMember` může vystupovat v pozici rodiče dalších uzlů a každý uzel je reprezentován pouze jednou informací o IP adrese a DNS (`HostInfo`)

Proměnné v těchto třídách jsou pomocí Hibernate mapovány do databáze, takže programátor nemusí spravovat přímo databázi a pokládat na ni klasické dotazy. Hibernate je mapovací nástroj pro platformu Java, který je schopen zajistit napojení na databázi pro objektově orientované programování.

## 7.2. Model aplikace

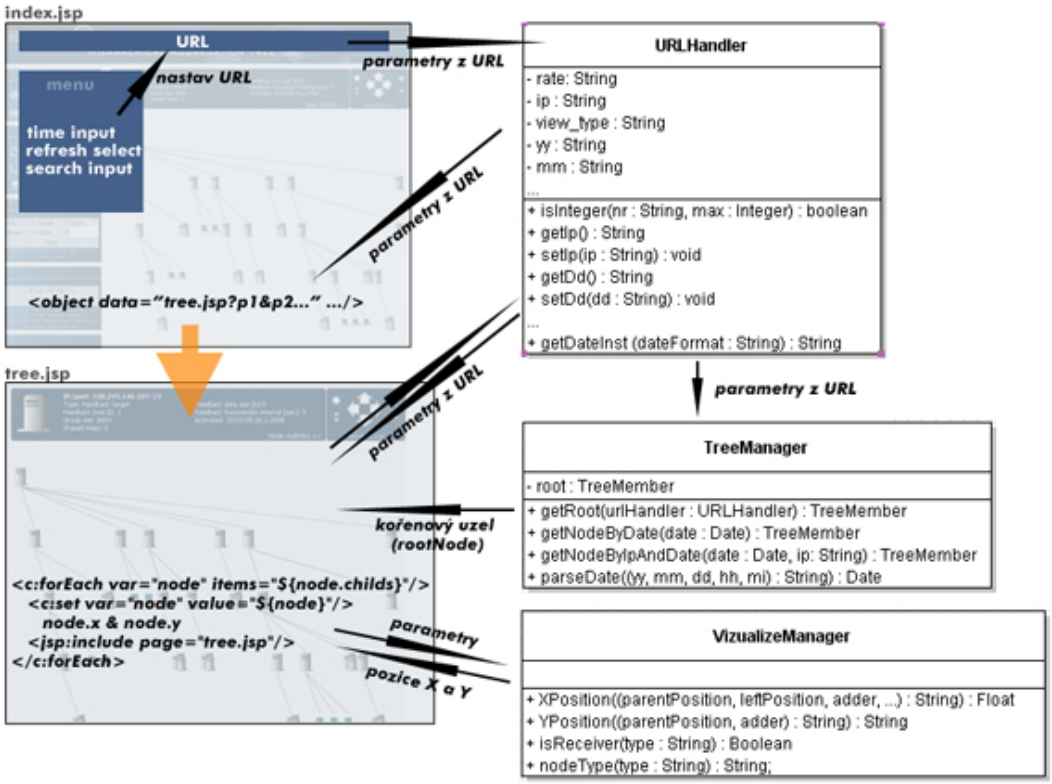
S ohledem na možnosti SVG a zejména na to, že není možné použít SVG spolu s HTML v rámci jedné webové stránky, zejména kvůli špatné podpoře v prohlížeči Microsoft Internet Explorer (viz výše) byla navržena struktura, kdy je dynamicky vygenerovaná vizualizace stromu vložena externě.

```
<!--[if IE]>
```

```
 <embed id="svg1" src="${treeURL}" pluginspage="http://www.adobe.com/svg/viewer/install/"
 type="image/svg+xml" width="1000" height="1000"> </embed>
<![endif]-->
<!--[if !IE]> -->
 <div class="svg" id="svg1" name="svg1"><object data="${treeURL}"
        type="image/svg+xml" width="1000" height="1000"></object></div>
<!-- <![endif]-->
```

Na výpisu zdrojového kódu výše je demonstrováno vložení SVG grafiky pomocí standardních tagů HTML. Je jí ovšem také nutné předat parametry pro vizualizaci, zde se opět zavolají metody třídy `URLHandler`.

Zjednodušený model aplikace je naznačen na následujícím obrázku. Úkolem třídy `TreeManager` je poskytnout rozhraní mezi vizualizací a samotným modelem stromu. Na základě informací, které má z adresní řádky (tedy z objektu `URLHandler`) může pokládat správné dotazy na databázi a vracet potřebná data.




**Obrázek 7.2. Zjednodušený diagram modelu aplikace a jeho funkce**

V JSP skriptu je tedy vytvořena nová instance třídy `URLHandler`, jež je naplněna daty z URL adresy.

```
<jsp:useBean id="URLHandler" scope="request" class="cz.vutbr.burgetrm.manager.URLHandler" />
 <%-- sets all url parameters --%>
<jsp:setProperty name="URLHandler" property="*" />
```

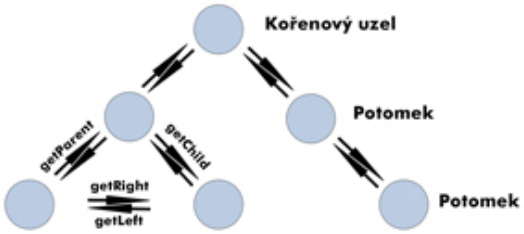
Vytvoří se také nová instance třídy `TreeManager` a zavolána metoda pro získání kořenového uzlu pro vizualizaci, jako parametr je předán `URLHandler`, aby bylo zajištěno vybrání dat dle správných parametrů:

```
<%
 TreeManager Tmng2 = new TreeManager();
 TreeMember node = Tmng2.getRoot(URLHandler);
 request.setAttribute("node", node); //sets object to JSTL
%>
<c:set var="node" value="${node}" scope="request"/>
```

Proměnná `node` typu `TreeMember` obsahuje tedy jeden uzel stromu a má k dispozici všechny metody nutné pro vizualizaci. Z obrázku je též patrna funkce třídy `VizualizeManager`, která na základě parametrů uzlů sousedících k danému uzlu vyřeší jeho pozici na plátně vizualizace.

## 7.3. Algoritmus vykreslení

Při návrhu algoritmu bylo vycházeno z toho, že díky modelu stromu jsou známy všechny vazby mezi uzly. Pracuje tedy na následujícím principu:




**Obrázek 7.3. Relace mezi uzly v síti**

1. Vyber kořenový uzel. Ten je určen tak, že metoda `getParent` vrací `NULL`, tedy daný uzel nemá žádného rodiče. Alternativou je, pokud uživatel chce zobrazit určitý podstrom. V tom případě je hodnota kořenového uzlu nastavena dle zadané IP adresy. Obdobně je to také s časovou informací - pokud chce uživatel zobrazit strom časově zpětně, budou z databáze vybrána data s příslušnou časovou značkou.

2. Vyber potomka a urči jeho polohu ( `getChild()` ).

3. Opakuj bod 2 dokud existují potomci (`getChild()!=NULL`).

4. Pokud neexistuje potomek, vrať se o úroveň výše a urči dalšího potomka.

Z výše uvedeného tedy plyne, že jsou při vykreslení rekurzivně volány funkce pro určení a výpočet polohy prvku ve stromu a ten je postupně vykreslován (metoda procházení stromem PRE-ORDER). Princip výpočtu přesné polohy na plátně vizualizace je založen opět na znalosti polohy nadřazeného uzlu a uzlu na levé straně:

1. Pro kořenový uzel zapiš polohu `[0;0]`.

2. Pro každý následující uzel zjisti polohu sousedícího uzlu nalevo.

    1. Pokud existuje, připiš k jeho pozici na souřadnici `x` konstantní hodnotu (`parent.x + C`). Pro pozici na souřadnici `y` zjisti, v jaké úrovni stromu stromu (v kolikátém stupni iterace) se nacházíme a vynásob ji konstantou (`status * K`). Hodnotu souřadnice `x` zapiš také do proměnné pro maximální hodnotu (`node.x = max_x`).

    2. Pokud neexistuje uzel nalevo, souřadnice `x` je součtem maximální hodnoty souřadnice dosud zapsané a konstanty (`max_x + C`). Hodnota na souřadnici y je opět dána stupněm iterace.

```
public static float x_position(String parent_position,
  String left_position, String x, String first, String last, String adder, String max_x) {
  float parent_pos = 0, left_pos = 0, add = 0, position=0, maximum_x=0;

  if (adder!=""){
   add = Float.valueOf(adder);
  }
  if (max_x!=""){
   maximum_x = Float.valueOf(max_x);
  }
  else maximum_x = 0;
  if (parent_position != "") { //if exist parent set position on paremt plus adder
   parent_pos = Float.valueOf(parent_position);
   position = maximum_x+add;
  }
  if (first == "true") {
   // reduce distance
  }
  if (left_position != "") {
   left_pos = Float.valueOf(left_position);
```

```
   position = left_pos + add;
   if (maximum_x>position) position = maximum_x;
  }
  if (first == "false" && last == "true") {
   // increase distance
  }
 return position;
}
```

Takto je zajištěno optimální vykreslení uzlů na plátno. Důraz byl kladen na to, že předpokládáme velmi velké množství uzlů k vykreslení, resp. menší množství sumarizačních uzlů, ale velmi velký počet uzlů koncových. Je také zřejmé, že jsme velmi omezeni velikostí zobrazovací jednotky (monitoru, televize), proto byl systém navržen tak, že koncové uzly jsou seskupeny pod jeden symbol na plátně. Jeho rozkliknutím dojde v detailnímu výpisu koncových uzlů.

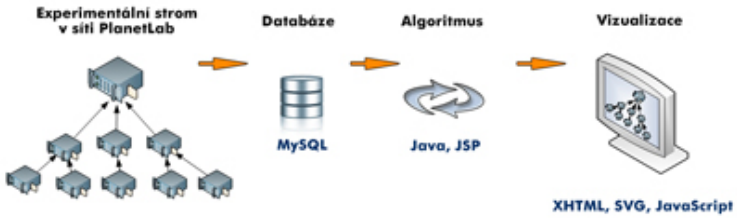
Zdrojové kódy pro výpočet pozice a procházení stromem jsou uvedeny v Příloze A.

# 8. Popis aplikace a jejích možností

Před vlastním popisem webové aplikace a jejích možností si shrňme, jak dříve popsané části projektu spolupracují.

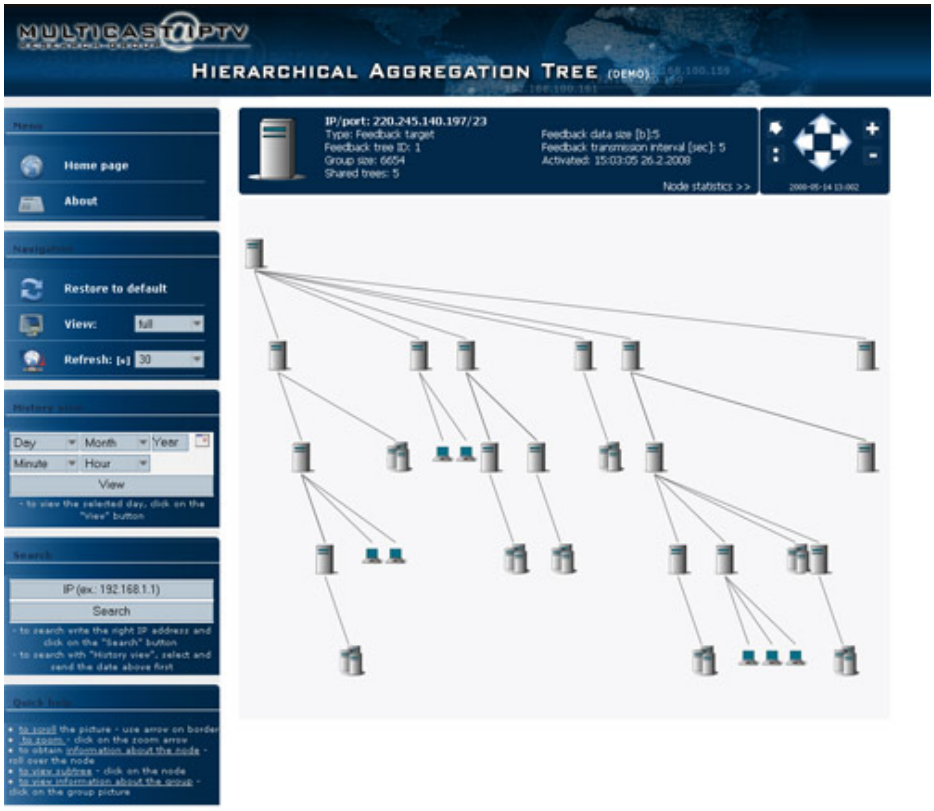
Na základě uzlů v experimentální síti PlanetLab a dříve popsaného protokolu TTL je složen strom pro zpracování dat od uživatelů IP televize. Informace o vzhledu stromu a parametrech jednotlivých uzlů (IP adresa, velikost odesílaných dat atd.) jsou v určitých časových intervalech ukládány do databáze MySQL. Data jsou načítána z databáze dle zadaných parametrů, například čas, od kterého chceme strom vykreslit, část podstromu atd. Vhodně zvolený algortimus v prostředí Java data z databáze načte a rekonstruuje strom vypočtením pozic jednotlivých uzlů a zavedením vzájemných vazeb mezi uzly (rodič, potomek). Výsledná data z algoritmu jsou přebírána servletem v JSP a pomocí nich je generována HTML stránka s vloženou vygenerovanou SVG grafikou.

Princip je možné popsat i z druhé stránky, od klienta. Ten při otevírání JSP stránky odesílá požadavek na server. Ten obdrží požadavek a přepošle jej na konkrétní servlet stroj. Stroj na základě algoritmu a dat z databáze o uzlech v síti PlanetLab vygeneruje odpověď (HTML a SVG) a zašle ji zpět klientovi, kterému je výsledná stránka zobrazena.




**Obrázek 8.1. Zjednodušený princip vizualizace dat**

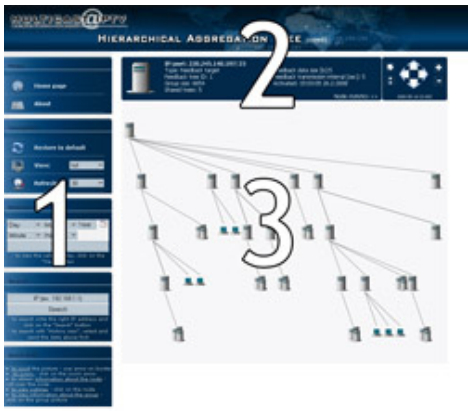
Výsledkem je tedy webová aplikace založená na technologiích XHTML, SVG a Javascriptu, která pomocí JSP dynamicky načítá data z databáze, zpracuje je a vyobrazí.




**Obrázek 8.2. Vzhled aplikace**

## 8.1. Vzhled aplikace a možnosti nastavení

Z hlediska požadavků na přehlednost, byla aplikace rozdělena na tři části (viz následující obrázek), na část s možnostmi nastavení (1), část navigace a základních informací o uzlu (2) a samotnou vizualizaci stromu (3). Není třeba do detailů popisovat vzhled a funkce každého prvku, budou zde tedy zmíněny jen ty zásadní.




**Obrázek 8.3. Části aplikace**

V levé části navigace se jedná zejména o nastavení kvality statických parametrů vizualizace:

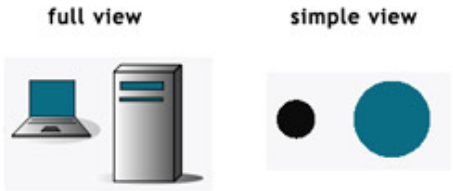
### View - typ zobrazení

Empiricky bylo zjištěno, že při velmi velkém počtu uzlů má aplikace vizualizace velmi pomalou odezvu. Je to dáno zejména tím, že až samotný prohlížeč se stará o vykreslení grafiky a pokud má vykreslovat mnoho grafických prvků je odezva logicky pomalejší i přes to, že grafické prvky vizualice byly voleny zejména s důrazem na rychlost a jednoduchost vykreslení. Velký podíl na snížení rychlosti má zejména vykreslování čar spojujících jednotlivé uzly. I samotný obrázek uzlu se skládá z několika grafických prvků - několik obdélníků vyplněných barevným přechodem, je tedy třeba brát v úvahu fakt, že prohlížeč musí interpretovat SVG tagy do vektorových obrazců a ty potom rasterizuje na plochu.

Vykreslovací model SVG ovšem nabízí možnost nastavení kvality zobrazení jak tvarů, tak textu a barev. K dispozici jsou možnosti pro vykreslení s ohledem na optimalizaci pro geometrickou preciznost, s důrazem na anti-aliasingové techniky nebo s ohledem na optimalizaci pro rychlost. Nutno podotknout, že o vlastnostech jednotlivého typu zobrazení si rozhoduje sám prohlížeč.

Aby bylo možné vizualizaci zobrazit i na pomalejších strojích, dáváme uživateli možnost nastavit si druh prohlížení. Pokud nastaví "simple view", k zobrazení budou použity velmi jednoduché geometrické tvary navíc se sníženou kvalitou zobrazení (`OptimizeSpeed`):

```
<g id="feedback_target" transform="scale(0.4) translate(0,0)" shape-rendering="optimizeSpeed">
 <circle r="30" fill="#0B6D84"/>
</g>
```




**Obrázek 8.4. Typy zobrazení uzlů**

### Refresh - čas obnovení

Jak již bylo zmíněno dříve, kvůli dynamice zobrazení bylo nutné zavést obnovení stránky. To může působit rušivě, pokud uživateli nezáleží na tom, aby vizualizace byla zcela aktuální. Předpokládá se, že vzhledem k prodlevám

získávání informací o uzlech bude implicitně nastavené obnovení po 30s dostačující. Uživatel má možnost nastavit si obnovení po 10, 30, 60s a obnovení vyplnout úplně.

### History view

V zadání práce byl kladen požadavek na možnost zobrazování stromu zpětně v čase, proto byl do aplikace přidán kalendář na možnost zadání data. Jako kalendáře je využito externí Javascript aplikace Unobtrusive Date-Picker Widget Update[1].

### Search

Zadání IP adresy je možno vyhledávat uzly v databázi. Odesláním IP adresy dojde k vyhledání a pokud je adresa nalezena, hledaný uzel je přiřazen na pomyslný vrchol stromu ve vizualizaci. Dojde tedy k vykreslení části stromu, která přísluší danému hledanému uzlu.

### Paramtery aplikace

V předchozí části zmíněné nastavení paramterů aplikace je realizováno HTML formuláři spolu s Javascriptem. Po změně nebo odeslání nějakého požadavku si hodnoty převezme javascriptová funkce, která se postará o automatické přesměrování stránky na novou s aktualizovanými parametry.

```
function refreshRate(){
 var sel2 = document.rate.refreshTime.selectedIndex;
 var val2 = document.rate.refreshTime.options[sel2].value;
 location.replace('${dateURL}ip=${URLHandler.ip}&gip=${URLHandler.gip}&
                  view_type=${URLHandler.view_type}&rate='+val2);
}
```

Parametry pro stránky se odesílají metodou GET. Ta spočívá v tom, že data jsou odesílána v URL adrese. Jednotlivé parametry jsou od samotné adresy stránky odděleny znakem "?" a mezi sebou odděleny znakem "&". V HTML lze data odesílat ještě metodou POST, ta předává proměnné v HTML hlavičkách, tudíž je nelze vidět v URL adrese. Tato metoda je spíše vhodnější pro přenos většího množství dat nebo pro "skryté" proměnné z formulářů.

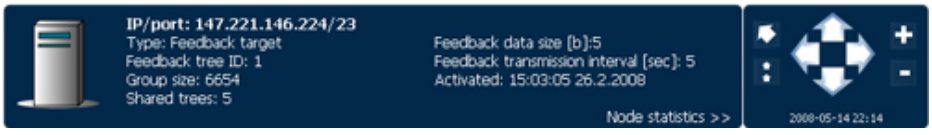
Díky tomu, že metoda GET předává parametry v URL adrese, je možné se vždy odkazovat na konkrétní typ zobrazení v určitém čase a například adresu přeposlat jinému uživateli, kterému bude zobrazena přesně stejná část stromu ve stejném čase. Adresa URL s parametry tedy vypadá takto:

```
http://adela.utko.feec.vutbr.cz:8080/TreeVisualization/index.jsp?
ip=147.221.146.224&view_type=simple&rate=10&mi=39&hh=21&mm=05&dd=14&yy=2008
```

Po řade jsou proměnné: IP adresa, typ zobrazení, obnovovací frekvence, minuty, hodiny, měsíc a rok pro zobrazení zpětně v čase.

## 8.2. Možnosti ovládání aplikace - navigační pole

V horní části aplikace je umístěno pole s informacemi o uzlu, který je nejvýše ve struktuře stromu vizualizace. Obsahuje například IP adresu uzlu, port, čas aktivace uzlu v síti apod. Vedle infomací je zde umístěn i odkaz na statistiky uzlu. Po kliknutí na tento odkaz budou v novém okně zobrazeny grafy spolehlivosti a vytíženosti uzlu v čase. Tato část je řešena v rámci jiné části projektu a přímo se nevztahuje k této práci.




**Obrázek 8.5. Informační a navigační pole**

Vedle Informačního pole je umístěno pole s ovládáním samotného vizualizačního plátna. Funkce panelu jsou snad zřejmé, nicméně je vhodné vysvětlení prinicipu jednotlivých funkcí.

[1] http://www.frequency-decoder.com/2006/10/02/unobtrusive-date-picker-widgit-update/

- Pohyb nad plátnem. Navigační kříž umožňuje pohyb nad plátnem čtyřmi směry. Tento pohyb je opět realizován Javascriptem. Ten může přistupovat k jednotlivým objektům struktury SVG (SVG DOM) a manipulovat s jejich vlastnostmi. Takto je manipulováno s vlastností `transform="translate(x y)"` plátna vizualice. Javascript funkce je spuštěna po najetí myši nad navigační šipku a v daných intervalech (pomocí časovačem volané funkce) provádí posouvání plátna v `x` a `y` souřadnicích.

- Zooming. Kliknutím na tlačítka "+" a "-" může uživatel zvětšovat a zmenšovat objekty na plátně. Tato funkce je využitelná zejména pokud se jedná o velké množství uzlů a vizualizace by tak mohla ztrácet na přehlednosti. Její princip je obdobný jako u předchozího. Javascript přistupuje k vlastnosti `viewbox`. Toto je velmi užitečná vlastnost SVG a umožňuje do plátna o velikosti `x` a `y` vměstnat objekt o velikosti `a` a `b` tak, že dle jeho velikosti ho automaticky zmenší nebo zvětší.

- O úroveň výše a o zobrazení zpět. Funkce, které umožní uživateli návrat na předchozí zobrazení nebo o jednu úroveň výše.

## 8.3. Možnosti ovládání aplikace - "plátno"

Zobrazovací plátno slouží k samotné vizualizaci stromu na základě experimentální sítě PlanetLab. Zde jsou tedy vykreslovány tři typy objektů - sumarizační uzly, klienti (přijímače) a skupiny přijímačů. Skupiny jsou zavedeny z toho důvodu, že se předpokládá velmi vysoký počet koncových uzlů, v řádu tisíců. Takový počet by se jednak nevešel na obrazovku a jednak by zcela jistě vedl k "zamrznutí" vizualizace, protože prohlížeč by musel zpracovávat velmi velké množství vektorové grafiky.

Funkce zobrazení informačního panelu byla rozebrána již dříve, nicméně zobrazení v SVG nabízí ještě další stupně interaktivity, například možnost vložit "pod objekt" odkaz. Takto je zajištěno, že pokud uživatel klikne na kořenový uzel ve vizualizaci, dostane se ve stromu o úroveň výše a například kliknutím na skupinu příjemců je zobrazen pohled na jednotlivé prvky skupiny s detailními informacemi.

# 9. Závěr

Úkolem této práce bylo vytvořit funkční systém pro vizualizaci relací mezi počítači na internetu za praktického otestování v síti PlanetLab. V práci byly diskutovány možné metody zobrazení, přičemž byla vybrána dle mého názoru metoda k vizualizaci na internetu nejvhodnější, tedy metoda, která přímo využívá webových technologií tak, aby byl výsledek dostupný odkudkoli a pokud možno s co nejmenšími "uživatelskými obtížemi".

Využívá se tedy poměrně nové technologie Scalable Vector Graphics (SVG), která je v současné době rychle se rozvíjející grafický formát zakládající se na vektorové grafice a vypadá to, že se zanedlouho stane vedoucím formátem na poli nejen webové grafiky, ale spolu s XML, CSS a XHTML univerzálním formátem pro výměnu grafických informací.

Velmi silným nástrojem je také možnost využívání ECMA skriptů (JavaScript) v SVG, čímž lze docílit vysokého stupně interaktivity obrázků dynamickou změnou jeho vlastností. Takto lze například přeskupovat jednotlivé objekty a řídit jejich vlastnosti v návaznosti na události generované uživatelem. Skripty lze spolu s vyplňováním, maskováním, efekty a animacemi různě kombinovat, a tak vytvářet velmi působivou interaktivní grafiku, kterou lze díky jejím malým nárokům na datový objem aplikovat téměř kdekoli v informačních technologiích (web, přenosná zařízení, mobilní telefony apod.).

S využitím již dříve získaných poznatků o podpoře v jednotlivých prohlížečích a aplikacích byl v rámci této práce vytvořen konkrétní a funkční návrh zobrazení včetně plné interaktivity. Spatřuji sice nemalou slabinu tohoto formátu v podpoře v prohlížečích, ale grafika byla uzpůsobena tak, aby bylo možné ji kvalitně zobrazit v různých prostředích.

Jako zdroj dat pro vizualizaci byla použita databáze MySQL s vloženými daty podle sítě PlanetLab. Prozatím nebylo možné dostávat do databáze výsledky z jiných větví projektu, jehož je tato práce součástí, proto jsou v databázi pouze testovací data.

Samozřejmě i tato metoda s sebou nese problémy. Díky tomu, že generujeme zdrojový kód SVG pomocí JSP a databáze, je poměrně jednoduché vytvořit velmi rozsáhlou grafiku s množstvím interaktivních funkcí, aniž by byla nějak náročná na datový přenos mezi serverem a webovým prohlížečem. Problém ovšem nastává v samotném prohlížeči, který grafiku na základě textových informací vykresluje. Při množství uzlů cca 700, kdy je na vykreslení každého uzlu použito několik čar, barevný přechod a řízení JavaScriptem, je pro klientský prohlížeč (počítač) poměrně náročné vykreslení takového množství vektorové grafiky a její následné převedení na rastr. Problémem je i zajištění dynamiky vykreslování, aby vizualizace byla schopna reagovat na změny uspořádání stromu.

Výsledek práce je k dipozici na webové stránce projektu[1], výňatky ze zdrojových kódů a ukázky výsledku v příloze této práce. Kompletní zdrojové kódy také na CD přiloženému k této práci.

Celý tento dokument je s využitím znalostí z předchozích prací zpracován v Docbooku, což sebou přináší mnohé výhody (např. výstup do několika formátů), ale také problémy spojené zejména s přípravou dokumentu k tisku.

[1] http://adela.utko.feec.vutbr.cz:8080/TreeVisualization/index.jsp

# Seznam literatury


[1] ARMSTRONG, Eric. *The J2EE 1.4 Tutorial* , 2005 [cit. 2008-05-20]. Dostupný z WWW:http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html

[2] BRANICKÝ, Marek. *ZJavaServer Pages pro všechny* [online]. Interval.cz. 6. 8. 2002 [cit. 2008-04-08]. Dostupný z WWW:http://interval.cz/clanky/javaserver-pages-pro-vsechny/. ISSN 1212-8651.

[3] BŘÍZA, Petr. *Znakové sady v praxi - UTF-8* [online]. Interval.cz. 30. 12. 2003 [cit. 2008-04-08]. Dostupný z WWW:http://interval.cz/clanky/znakove-sady-v-praxi-utf-8/. ISSN 1212-8651.

[4] BURGET, Radim, KOMOSNÝ, Dan.*Realtime control protocol and its improvements for Internet Protocol Television*, International Transaction on Computer Science and Engineering, 2006. roč. 2006 [cit. 2008-05-01], č. 31, s. 1 - 12. Dostupný z WWW: http://adela.utko.feec.vutbr.cz/projects/images/pdf_files/rtcp_improvements_iptv.pdf?url=http%3A//adela.utko.feec.vutbr.cz/projects/images/pdf_files/rtcp_improvements_iptv.pdf&id=3&pagename=FILE:rtcp_improvements_iptv.pdf. ISSN 1738-6438.

[5] CIMBÁLEK, Přemysl. *Vektorový grafický formát SVG*, 2007. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií. Bakalářská práce.

[6] CIMBÁLEK, Přemysl. *Semestrální projekt 2*, 2006. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií. Semestrální práce.

[7] *DOM - Document Object Model* [online]. Tvorba-webu.cz. [cit. 2006-06-04]. Dostupný z WWW: http://www.tvorba-webu.cz/DOM.php3.

[8] Extensible Markup Language (XML) 1.0 (Third Edition), http://www.w3.org/TR/2004/REC-xml-20040204

[9] HEJRAL, Martin.*Průvodce SVG* [online]. Interval.cz. 15.4.2003 [cit. 2006-06-04]. Dostupný z WWW: <?xml version="1.0"?> <ns:clipboard xmlns:ns="http://www.xmlmind.com/xmleditor/namespace/clipboard" ><bibliomisc >http://interval.cz/clanky/pruvodce-svg-hlaseni-o-stavu-vyvoje/. </bibliomisc ><issn >ISSN 1212-8651</issn ></ns:clipboard >.

[10] HEJRAL, Martin. *Kurz SVG* [online]. Interval.cz. 7.7.2004 [cit. 2006-06-04]. Dostupný z WWW: http://interval.cz/clanky/kurz-svg-struktura-dokumentu-zobrazovaci-a-vykreslovaci-model/. ISSN 1212-8651.

[11] KOMOSNÝ, Dan. *Nové směry vývoje protokolu RTP/RTCP pro rozsáhlé konference v Internetu* [online]. 27.10.2004. roč. 2004 [cit. 2008-05-01], č. 52. Dostupný z WWW: http://www.elektrorevue.cz/clanky/04052/index.html#2

[12] KOMOSNÝ Dan, NOVOTNÝ Vít. *Tree Structure for Source-Specific Multicast with feedback Aggregation*, ICN07 - The Sixth International Conference on Networking, Martinique, 2007 [cit. 2008-05-01]. Dostupný z WWW: http://adela.utko.feec.vutbr.cz/projects/images/pdf_files/rtcp_improvements_iptv.pdf?url=http%3A//adela.utko.feec.vutbr.cz/projects/images/pdf_files/rtcp_improvements_iptv.pdf&id=3&pagename=FILE:rtcp_improvements_iptv.pdf. ISBN 0-7695-2805-8.

[13] KOSEK, Jiří.*Tvorba webových stránek a aplikací* [online]. IZI228. 5.5.2006 [cit. 2006-06-04]. Dostupný z WWW: http://badame.vse.cz/izi228/.

[14] KUČEROVÁ, Kateřina. EPG - užitečný pomocník nebo zbytečná funkce? [online]. 28.02.2007 [cit. 2008-02-04]. Dostupný z WWW: http://www.rozhlas.cz/digital/prijimace/_zprava/323948

[15] MAIXNER, David. *Java Servlety, JSP a webové servery s jejich podporou*, 2002. Brno: Masarykova Univerzita, Fakulta informatiky. Bakalářská práce.

[16] MARTIN, Davis. *ITU-T SG15 Ethernet Frame Size Issue*, ICN07 - The Sixth International Conference on Networking, Martinique, 2004 [cit. 2008-05-01]. Dostupný z WWW: www.ieee802.org/3/frame_study/public/0407/martin_ar_1_0407.pdf

[17] *Mobile SVG: The Graphics Wave of the Future*[online]. Ikivo. 2005 [cit. 2006-06-04]. Dostupný z WWW: http://whitepapers.zdnet.co.uk/0,1000000651,260164955p,00.htm

[18] *Mozilla v češtině* [online]. Czilla. 2006 [cit. 2006-05-10]. Dostupný z WWW: http://www.czilla.cz. ISSN 1212-8309.

[19] NAVRÁTIL, J. *PlanetLab - model budoucího Internetu.* Zpravodaj ÚVT MU. Guadeloupe, 2007. [cit. 2008-05-05], 2006, roč. XVI, č. 5, s. 1-5. Dostupný z WWW: http://www.ics.muni.cz/zpravodaj/articles/525.html. ISSN 1212-0901

[20] NOVOTNÝ, Vít, KOMOSNÝ, Dan. *Optimization of Large-Scale RTCP Feedback Reporting*, ICWMC 2007. ICWMC 2007 - The Third International Conference on Wireless and Mobile Communications. Guadeloupe, 2007 [cit. 2008-05-05]. Dostupný z WWW: http://www.academypublisher.com/jnw/vol03/no03/jnw03030110.pdf. ISBN 0-7695-2796-5.

[21] Opera Software ASA.*Opera Software* [online]. Opera. 2006 [cit. 2006-06-04]. Dostupný z WWW: <? xml version="1.0"?> <ns:clipboard xmlns:ns="http://www.xmlmind.com/xmleditor/namespace/clipboard" ><bibliomisc >http://</bibliomisc >w<issn >ww.opera.com</issn ></ns:clipboard >.

[22] PETERKA, Jiří. Jak funguje IPTV? Lupa.cz. [online]. 24. 8. 2006 [cit. 2008-02-04]. Dostupný z WWW: http://www.lupa.cz/clanky/jak-funguje-iptv/ . ISSN 1213-0702.

[23] PETERKA, Jiří. Problém poslední míle a zpřístupnění místní smyčky [online]. [cit. 2008-02-04]. Dostupný z WWW: http://www.earchiv.cz/b03/b0500001.php3

[24] PUŽMANOVÁ, Rita. HSDPA do sítí 3G Dsl.cz. [online]. 29. 11. 2005 [cit. 2008-02-04]. Dostupný z WWW: http://www.dsl.cz/clanky-dsl/clanek-283/HSDPA-do-siti-3G

[25] PUŽMANOVÁ, Rita. Streaming media (4): transportní protokoly RTP/RTCP Dsl.cz. [online]. 18. 10. 2004 [cit. 2008-02-04]. Dostupný z WWW: http://www.dsl.cz/clanky-dsl/clanek-60/Streaming-media-(4):-transportni-protokoly-RTP-RTCP

[26] PIŠTĚK, Petr. *Multicast: skupinové vysílání*, Zpravodaj ÚVT MU, 1998. roč. VIII [cit. 2008-05-01], č. 5, s. 13 - 15. Dostupný z WWW: http://www.ics.muni.cz/zpravodaj/articles/134.html. ISSN 1212-0901.

[27] ROZSYPAL, Petr. *Metody POST a GET* [online]. Interval.cz. 1. 7. 2006 [cit. 2006-06-04]. Dostupný z WWW:http://php.interval.cz/clanky/metody-post-a-get/. ISSN 1212-8651.

[28] ŘÍČNÝ, Václav, KRATOCHVÍL, Tomáš.*Základy televizní techniky*, 2004. 14 s. Fakulta elektrotechniky a komunikačních technologií, VUT. Skriptum. . ISSN 1738-6438.

[29] Scalable Vector Graphics (SVG) 1.1 Specification, http://www.w3.org/TR/2003/REC-SVG11-20030114

[30] Služby Video on Demand a Pay per View Digizone.cz. [online]. [cit. 2008-02-04]. Dostupný z WWW: http://tutorialy.digizone.cz/digitalni-obsah/video-on-demand-pay-per-view/ ISSN 1801-4933.

[31] STANÍČEK, Petr. *Proč používám XHTML* [online]. Interval.cz. 14.10.2004 [cit. 2007-12-04]. Dostupný z WWW: http://interval.cz/clanky/proc-pouzivam-xhtml/. ISSN 1212-8651.

[32] ŠŤASTNÝ, Petr. *Typy tabulek v MySQL* [online]. 27.03.2007 [cit. 2008-04-08]. Dostupný z WWW:http://www.pweb.cz/a/14/typy-tabulek-v-mysql.html.

[33] VÁCLAVEK, Petr. *Seriál JavaScript pro začátečníky - 1. díl* [online]. 9.09.05 [cit. 2008-04-08]. Dostupný z WWW:http://petr.vaclavek.com/article/277/Javascript-pro-zacatecniky-serial

# Rejstřík

# Příloha A. Zdrojové kódy a ukázky řešení vizualizace

V této části jsou uvedeny ukázky výsledku práce. Z důvodu rozsahu není možné uvést kompletní zdrojové kódy, proto budou vybrány pouze důležité části.

## A.1. SVG

### A.1.1. Předpřipravené prvky

K vykreslení bylo třeba předpřipravit prvky vizualizace, a to obrázek serveru a klienta, které budou poté opakovaně vkládány do celkového řešení vizualizace. V SVG je samozřejmě možné vkládat externí obrázky v rastrových formátech, nicméně kvůli možným problémům například se zvětšováním bylo zvoleno opět SVG. Jeho výhoda spočívá v tom, že kvalita obrázku nebude se zvětšením do určité míry změněna, vektory jsou totiž při každé změně velikosti vypočítávány znovu.

**Obrázek A.1. Obrázek prvku "klient"**

Zdrojový kód není třeba komentovat, je poměrně srozumitelný. Jeho základem jsou pouze křivky, jež mají definovanou výplň. V případě "monitoru" je to prostá barva, v případě "těla notebooku" se jedná o barevný přechod.

```
<g id="client" transform="scale(0.15) translate(-45,-50)" onmouseout="zmizet()">
 <linearGradient id="XMLID_44_" gradientUnits="userSpaceOnUse" x1="0" y1="51.75" x2="95"
 y2="51.75">
  <stop offset="0" style="stop-color:#FFFFFF"/>
  <stop offset="1" style="stop-color:#565656"/>
 </linearGradient>
 <ellipse opacity="0.3" fill="url(#XMLID_44_)" cx="47.5" cy="51.75" rx="47.5" ry="14.75"/>
 <linearGradient id="XMLID_55_" gradientUnits="userSpaceOnUse" x1="3" y1="58.3359"
 x2="84.3779" y2="58.3359">
  <stop offset="0" style="stop-color:#FFFFFF"/>
  <stop offset="1" style="stop-color:#565656"/>
 </linearGradient>
 <rect x="3" y="57.583" fill="url(#XMLID_55_)" stroke="#000000" width="81.378" height="1.506"/
>
 <rect x="21.333" y="0.5" fill="#0B6D84" stroke="#000000" width="47.001" height="38.001"/>
 <line fill="none" stroke="#000000" x1="21.25" y1="38.625" x2="68.375" y2="38.625"/>
 <linearGradient id="XMLID_66_" gradientUnits="userSpaceOnUse" x1="3.125" y1="48" x2="84.375"
 y2="48">
  <stop offset="0" style="stop-color:#FFFFFF"/>
  <stop offset="1" style="stop-color:#565656"/>
 </linearGradient>
 <polyline fill="url(#XMLID_66_)" stroke="#000000" points="68.375,38.625 84.375,57.25
 84.25,57.25 3.125,57.25 3.125,57.375
  21.25,38.625 "/>
 <line fill="none" stroke="#000000" x1="22.667" y1="41.667" x2="66.5" y2="41.667"/>
 <line fill="none" stroke="#000000" x1="19.167" y1="44.667" x2="69.667" y2="44.667"/>
 <polyline fill="#3F3F3F" stroke="#000000" points="49.053,48.934 52.914,53.428 52.884,53.428
 33.307,53.428 33.307,53.458
  37.681,48.934 "/>
 <line fill="none" stroke="#000000" x1="37.333" y1="48.917" x2="49.458" y2="48.917"/>
</g>
```

Obrázek serveru je zpracován obdobným způsobem:

**Obrázek A.2. Obrázek prvku "server"**

```
<g id="server" transform="scale(0.25) translate(-45,-50)" onmouseout="zmizet()">
 <linearGradient id="XMLID_4_" gradientUnits="userSpaceOnUse" x1="0" y1="89.0791" x2="88"
 y2="89.0791">
  <stop offset="0" style="stop-color:#FFFFFF"/>
  <stop offset="1" style="stop-color:#565656"/>
 </linearGradient>
 <ellipse fill="url(#XMLID_4_)" cx="44" cy="89.079" rx="44" ry="4.75"/>
 <polygon fill="#FFFFFF" stroke="#000000" stroke-linecap="square" stroke-linejoin="bevel"
 points="70.625,86.253 65.125,91.753
  65.125,6.247 70.625,0.747 "/>
 <linearGradient id="XMLID_5_" gradientUnits="userSpaceOnUse" x1="18.0991" y1="48.832" x2="65"
 y2="48.832">
  <stop offset="0" style="stop-color:#FFFFFF"/>
  <stop offset="1" style="stop-color:#565656"/>
 </linearGradient>
 <rect x="18.099" y="6.079" fill="url(#XMLID_5_)" stroke="#000000" width="46.901"
 height="85.506"/>
 <rect x="23.25" y="14.079" fill="#0B6D84" stroke="#000000" width="33.25" height="5.75"/>
 <linearGradient id="XMLID_6_" gradientUnits="userSpaceOnUse" x1="17.917" y1="3.25"
 x2="70.418" y2="3.25">
  <stop offset="0" style="stop-color:#FFFFFF"/>
  <stop offset="1" style="stop-color:#5D6060"/>
 </linearGradient>
 <polygon fill="url(#XMLID_6_)" stroke="#000000" stroke-linecap="square" stroke-
linejoin="bevel" points="17.917,6 23.417,0.5
  70.418,0.5 64.918,6 "/>
 <polygon fill="#474747" stroke="#000000" stroke-linecap="square" stroke-linejoin="bevel"
 points="70.625,86.253 65.125,91.753
  65.125,6.247 70.625,0.747 "/>
 <rect x="22.75" y="26.329" fill="#0B6D84" stroke="#000000" width="33.25" height="2.5"/>
</g>
```

## A.1.2. Vygenerovaný zdrojový kód

Uveden je pouze zkrácený výňatek ze zdrojových kódů.

V úvodu lze vidět hlavičku xml dokumentu a typ dokumentu. Následuje fragment `<svg>`, v němž je obsažena samotná grafika. V první části grafiky jsou uvedeny definiční objekty, tedy objekty, které se v grafice opakují. Následuje kód pro informační panel a informace o samotném uzlu.

```
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11-flat-20030114.dtd">
<svg width="1000" height="1000" xmlns:xlink="http://www.w3.org/1999/xlink" xmlns="http://
www.w3.org/2000/svg">
  <script type="text/ecmascript" xlink:href="../scripts/effects.js"/>
  <script type="text/ecmascript" xlink:href="../scripts/infobox.js"/>
  <script type="text/ecmascript" xlink:href="../scripts/infobox_ft.js"/>
  <script type="text/ecmascript" xlink:href="../scripts/infobox_g.js"/>
  <script type="text/ecmascript" xlink:href="../scripts/infobox_rec.js"/>
  <script type="text/ecmascript" xlink:href="../scripts/navigations.js"/>
  <script type="text/ecmascript" xlink:href="../scripts/zooming.js"/>
  <defs>
  <linearGradient id="arrowXMLID_3_" gradientUnits="userSpaceOnUse" x1="0" y1="100" x2="32"
 y2="100">
    <stop  offset="0" style="stop-color:#FFFFFF"/>
    <stop  offset="1" style="stop-color:#0B4D96"/>
  </linearGradient>
  <g id="nav_arrow">
    <polygon id="arrow" fill="#FFFFFF" stroke="url(#arrowXMLID_3_)" stroke-width="3"
            points="0  0, 10 0,10 -5 , 20 7.5, 10 20, 10 15, 0 15" />
```

```
  </g>
  <g id="nav_zoom_plus" transform="translate(0 0) rotate(0)" font-size="20pt" >
  <rect id="nav_rect_plus"
        x="0" y="0" width="20" height="20"
        fill="#FFFFFF" stroke="url(#arrowXMLID_3_)" stroke-width="3" opacity=".05" >
    <title>Click to zoom in</title>
  </rect>
  <text dx="2" dy="19" font-weight="bold"   stroke="#02294a" fill="white" stroke-width="0">
  +
  </text>
  </g>
  <g id="nav_zoom_minus" transform="translate(0 0) rotate(0)" font-size="20pt">
    <rect id="nav_rect_minus"
        x="0" y="0" width="20" height="20"
        fill="#FFFFFF" stroke="url(#arrowXMLID_3_)" stroke-width="3"  opacity=".05"/>
    <text dx="6" dy="17" font-weight="bold" stroke="#FFFFFF" fill="white" stroke-width="0" >-
</text>
  </g>
  <g id="nav_back"  font-size="20pt">
    <g>
      <rect id="nav_rect_back" transform="translate(0 -27)"
        x="0" y="0" width="20" height="20"
        fill="#FFFFFF" stroke="url(#arrowXMLID_3_)" stroke-width="3"  opacity=".05"/>
      <polygon id="arrow" transform="translate(10 -10) rotate(-135) scale(.7)" fill="#FFFFFF"
 stroke="url(#arrowXMLID_3_)" stroke-width="0"
           points="0  0, 10 0,10 -5 , 20 7.5, 10 20, 10 15, 0 15" />
    </g>
  </g>
  <g id="nav_top"  font-size="20pt">
    <g>
      <rect id="nav_rect_top" transform="translate(0 -27)"
        x="0" y="0" width="20" height="20"
        fill="#FFFFFF" stroke="url(#arrowXMLID_3_)" stroke-width="3"  opacity=".05"/>

 <polygon id="arrow" transform="translate(7.5 -19) rotate(-90) scale(.3)" fill="#FFFFFF"
 stroke="url(#arrowXMLID_3_)" stroke-width="0"
           points="0  0, 10 0,10 -5 , 20 7.5, 10 20, 10 15, 0 15" />
      <polygon id="arrow" transform="translate(7.5 -9) rotate(-90) scale(.3)" fill="#FFFFFF"
 stroke="url(#arrowXMLID_3_)" stroke-width="0"
           points="0  0, 10 0,10 -5 , 20 7.5, 10 20, 10 15, 0 15" />
    </g>
  </g>
  <g id="nav_arrow2"  >
    <linearGradient id="arrowXMLID_33_" gradientUnits="userSpaceOnUse" x1="0" y1="100" x2="32"
 y2="100">
      <stop  offset="0" style="stop-color:#FFFFFF"/>
      <stop  offset="1" style="stop-color:#0B4D96"/>
    </linearGradient>
    <!-- <rect fill="url(#arrowXMLID_33_)" width="32" height="60" y="30"/> -->
    <g>
      <linearGradient id="arrowXMLID_44_" gradientUnits="userSpaceOnUse" x1="9.5" y1="98.2803"
 x2="26" y2="98.2803">
        <stop  offset="0" style="stop-color:#FFFFFF"/>
        <stop  offset="1" style="stop-color:#FFFFFF"/>
      </linearGradient>
      <polygon fill="url(#arrowXMLID_44_)" points="9.5,98.28 9.5,78.198 17.75,88.239 26,98.28
 17.75,108.322 9.5,118.363   "/>
      <g>
        <polygon fill="none" points="9.5,98.28 9.5,78.198 17.75,88.239 26,98.28 17.75,108.322
 9.5,118.363    "/>
        <path fill="url(#arrowXMLID_33_)" id="arrow2" d="M10.994,
 98.422c0-6.694,0-13.388,0-20.083c-0.879, 0.276-1.758,0.553-2.638,0.829
    c5.5,6.694,11,13.388,16.5,20.083c-0.027-0.61-0.056-1.221-0.084-1.831c-5.5,
 6.694-11,13.389-16.5,20.083
    c0.907,0.334,1.815,0.668,2.722,1.002C10.994,111.811,10.994,105.116,10.994,
 98.422c0-1.921-2.987-2.215-2.987-0.283
    c0,6.694,0,13.389,0, 20.083c0,1.52,1.737,2.201,2.722,1.002c5.5-6.694,
 11-13.389,16.5-20.083
    c0.429-0.521,
 0.319-1.339-0.084-1.831c-5.501-6.694-11-13.388-16.5-20.083c-0.774-0.942-2.638-0.502-2.638,
 0.829
    c0,6.694,0,13.388,0,20.083C8.007,100.06,10.994,100.354,10.994,98.422z"> </path>
```

```
      </g>
    </g>
  </g>
  </defs>
  <g transform="translate(20,0)">
    <g transform="translate(10,30)">
      <defs>
        <g id="feedback_target" transform="scale(0.4) translate(-45,-50)" >
          <linearGradient id="XMLID_4_" gradientUnits="userSpaceOnUse" x1="0" y1="89.0791"
 x2="88" y2="89.0791">
            <stop  offset="0" style="stop-color:#FFFFFF"/>
            <stop  offset="1" style="stop-color:#565656"/>
          </linearGradient>
          <ellipse fill="url(#XMLID_4_)" cx="44" cy="89.079" rx="44" ry="4.75"/>
          <polygon fill="#FFFFFF" stroke="#000000" stroke-linecap="square" stroke-
linejoin="bevel" points="70.625,86.253 65.125,91.753
 65.125,6.247 70.625,0.747 "/>
          <linearGradient id="XMLID_5_" gradientUnits="userSpaceOnUse" x1="18.0991"
 y1="48.832" x2="65" y2="48.832">
            <stop  offset="0" style="stop-color:#FFFFFF"/>
            <stop  offset="1" style="stop-color:#565656"/>
          </linearGradient>
          <rect x="18.099" y="6.079" fill="url(#XMLID_5_)" stroke="#000000" width="46.901"
 height="85.506"/>
          <rect x="23.25" y="14.079" fill="#0B6D84" stroke="#000000" width="33.25"
 height="5.75"/>
          <linearGradient id="XMLID_6_" gradientUnits="userSpaceOnUse" x1="17.917" y1="3.25"
 x2="70.418" y2="3.25">
            <stop  offset="0" style="stop-color:#FFFFFF"/>
            <stop  offset="1" style="stop-color:#5D6060"/>

          </linearGradient>
          <polygon fill="url(#XMLID_6_)" stroke="#000000" stroke-linecap="square" stroke-
linejoin="bevel" points="17.917,6 23.417,0.5
 70.418,0.5 64.918,6 "/>
          <polygon fill="#474747" stroke="#000000" stroke-linecap="square" stroke-
linejoin="bevel" points="70.625,86.253 65.125,91.753
 65.125,6.247 70.625,0.747 "/>
          <rect x="22.75" y="26.329" fill="#0B6D84" stroke="#000000" width="33.25"
 height="2.5"/>
        </g>
      </defs>
    </g>
    <g id="infobox" font-family="Tahoma" font-size="12">
      <rect width="599" rx="5" height="100" fill="#02294a" y="0" x="0"/>
      <use xlink:href="#feedback_target" transform="translate(45, 50) scale(1.9)" />
      <g transform="translate(100, 20)" fill="#FFFFFF" stroke-width="0" text-
rendering="optimizeLegibility">
        <text>
          <tspan font-weight="bold" >IP/port: 156.56.250.226/23</tspan>
        </text>
        <text>
          <tspan dy="15">Type: Feedback target</tspan>
        </text>
        <text>
          <tspan dy="30">Feedback tree ID: 1</tspan>
        </text>
        <text>
          <tspan dy="45">Group size: 6654</tspan>
        </text>
        <text>
          <tspan dy="60">Shared trees: 5</tspan>
        </text>
        <a xlink:href="../index.jsp?name=133.1.74.163" xlink:title="See node statistics"
 target="_blank">
        <text>
          <tspan dy="75" dx="390">Node statistics >></tspan>
        </text>
        </a>
        <text>
          <tspan dy="15" dx="250">Feedback data size [b]:5 </tspan>
        </text>
```

```
        <text>
          <tspan dy="30" dx="250">Feedback transmission interval [sec]: 5</tspan>
        </text>
        <text>
          <tspan dy="45" dx="250">Activated: 15:03:05 26.2.2008 </tspan>
        </text>
        <text>
          <tspan dy="60" dx="250"></tspan>
        </text>
      </g>
    </g>
    <g id="navigation" transform="translate(600, 0)">
      <rect width="150" rx="5" height="100" fill="#02294a" y="0" x="0"
 onmouseover="translate_view('clear')"/>

      <use xlink:href="#nav_arrow" id="nav_right"
   transform="translate(85, 32)" onmouseout="translate_view('clear')"
   onmouseover="translate_view('right')" xlink:title="Scroll right"/>
      <use xlink:href="#nav_arrow" id="nav_left"
   transform="matrix(-1,0,0,1,0,0) translate(-60, 32)"
   onmouseover="translate_view('left')"
   onmouseout="translate_view('clear')" xlink:title="Scroll left"/>
      <use xlink:href="#nav_arrow" id="nav_top"
   transform="rotate(-90) translate(-28, 65)"
   onmouseover="translate_view('top')"
   onmouseout="translate_view('clear')" xlink:title="Scroll up"/>
      <use xlink:href="#nav_arrow" id="nav_down"
   transform="rotate(90) translate(52, -80)"
   onmouseover="translate_view('down')"
   onmouseout="translate_view('clear')" xlink:title="Scroll down"/>
      <a xlink:href="javascript: history.go(-1)" target="_top" xlink:title="Go to the previous
 view">
      <use xlink:href="#nav_back" id="nav_back"
   transform="rotate(0) translate(9, 41)" onmouseover="highlight_in('nav_rect_back')"
 onmouseout="highlight_out('nav_rect_back')"></use>
      </a> <a xlink:href="../index.jsp?
name=133.1.74.163&amp;view_type=full&amp;rate=30&amp;date-sel2-mi=21&amp;date-sel2-
hh=14&amp;date-sel2-mm=05&amp;date-sel2-dd=12&amp;date-sel2=2008" target="_top"
 xlink:title="Go to the parent node">
      <use xlink:href="#nav_top"  id="nav_top" transform="rotate(0) translate(9, 71)"
 onmouseover="highlight_in('nav_rect_top')" onmouseout="highlight_out('nav_rect_top')"></use>
      </a>
      <use xlink:href="#nav_zoom_plus" id="zoom_in" xlink:title="Zoom in"
   transform="rotate(0) translate(120,15)" onclick="zoom_button(70)"
 onmouseover="highlight_in('nav_rect_plus')" onmouseout="highlight_out('nav_rect_plus')" />
      <use xlink:href="#nav_zoom_minus" id="zoom_out" xlink:title="Zoom out"
   transform="rotate(0) translate(120, 45)" onclick="zoom_button(-70)"
 onmouseover="highlight_in('nav_rect_minus')" onmouseout="highlight_out('nav_rect_minus')" />
      <g text-anchor="middle" font-family="Tahoma" font-size="10" fill="#dadada" stroke-
width="0" text-rendering="optimizeLegibility">
        <text transform="translate (75 95)">2008-05-12 14:21</text>
      </g>
    </g>
    <!--  <rect width="750" height="600" fill="#F5F6F7" y="105" x="0"
   opacity="0.85" onmouseup="catch_2(evt)" onmousedown="catch_1(evt)" />  -->
    <rect width="750" height="600" fill="#F5F6F7" y="105" x="0"
   opacity="0.85" />
    <svg id="nodes" width="750" height="600" x="0" y="100"
   viewBox="0 0 750 600" preserveAspectRatio="xMidYMid meet">
      <g id="nodes_group" transform="translate(20,40)"
    onmousedown="catch_1(evt)">
        <g id="node156.56.250.226"
     transform="translate(0.0,0.0)">
          <!--  <a xlink:href="javascript: history.go(-1)" target="_top"> -->
          <a xlink:href="../index.jsp?
name=133.1.74.163&amp;view_type=full&amp;rate=30&amp;date-sel2-mi=21&amp;date-sel2-
hh=14&amp;date-sel2-mm=05&amp;date-sel2-dd=12&amp;date-sel2=2008" target="_top">
          <use xlink:href="#feedback_target"
    id="x156.56.250.226"
    onmousemove="pozice_ft(evt, '156.56.250.226'), scale_in(evt, '156.56.250.226')"
    onmouseout="scale_out(evt, '156.56.250.226'), zmizet_ft()"> </use>
          <desc id="type156.56.250.226">trueFeedback target</desc>
```

```
          <desc id="ip156.56.250.226">156.56.250.226</desc>
          <desc id="tree156.56.250.226">1</desc>
          <desc id="gsize156.56.250.226">26</desc>
          <desc id="res156.56.250.226">90</desc>
          <desc id="shared_tree156.56.250.226">1</desc>
          <desc id="data156.56.250.226">6654</desc>
          <desc id="interval156.56.250.226">5</desc>
          <desc id="activated156.56.250.226">16:44:20 26.2.2008</desc>
          </a> </g>
      </g>

    </svg>
    <g id="extra_text" visibility="hidden" text-rendering="optimizeLegibility">
      <rect width="250" height="60" fill="lightsteelblue" y="-15"
    x="-10" opacity="0.85" stroke-width="1" stroke="#8AA3BC" />
      <text id="text_ip" font-size="14" font-weight="bold" y="0" x="0"> </text>
      <text id="text_typ" font-size="10" font-weight="bold" y="20"
    x="0"> </text>
      <text id="text_number" font-size="10" y="40" x="0"> </text>
    </g>
    <g id="extra_text_ft" visibility="hidden" font-family="Tahoma" text-
rendering="optimizeLegibility">
      <rect width="250" height="180" fill="lightsteelblue" y="-15"
    x="-10" opacity="0.85" stroke-width="1" stroke="#8AA3BC" />
      <text id="text_type_ft" font-size="10" font-weight="bold" y="0" x="0"> </text>
      <text id="text_ip_ft" font-size="14" font-weight="bold" y="20"
    x="0"> </text>
      <text id="text_tree_ft" font-size="10" y="40" x="0"> </text>
      <text id="text_gsize_ft" font-size="10" y="60" x="0"> </text>
      <text id="text_res_ft" font-size="10" y="80" x="0"> </text>
      <text id="text_shared_tree_ft" font-size="10" y="100" x="0"> </text>
      <text id="text_data_ft" font-size="10" y="120" x="0"> </text>
      <text id="text_interval_ft" font-size="10" y="140" x="0"> </text>
      <text id="text_activated_ft" font-size="10" y="160" x="0"> </text>
    </g>
    <g id="extra_text_rec" visibility="hidden" font-family="Tahoma" text-
rendering="optimizeLegibility">
      <rect width="250" height="180" fill="lightsteelblue" y="-15"
    x="-10" opacity="0.85" stroke-width="1" stroke="#8AA3BC" />
      <text id="text_type_rec" font-size="10" font-weight="bold" y="0" x="0"> </text>
      <text id="text_ip_rec" font-size="14" font-weight="bold" y="20"
    x="0"> </text>
      <text id="text_tree_rec" font-size="10" y="40" x="0"> </text>
      <text id="text_data_rec" font-size="10" y="120" x="0"> </text>
      <text id="text_interval_rec" font-size="10" y="140" x="0"> </text>
      <text id="text_activated_rec" font-size="10" y="160" x="0"> </text>
      <text id="text_loss_rec" font-size="10" y="60" x="0"> </text>
      <text id="text_delay_rec" font-size="10" y="80" x="0"> </text>
      <text id="text_jitter_rec" font-size="10" y="100" x="0"> </text>
    </g>
    <g id="extra_text_g" visibility="hidden" font-family="Tahoma" text-
rendering="optimizeLegibility">
      <rect width="250" height="60" fill="lightsteelblue" y="-15"
    x="-10" opacity="0.85" stroke-width="1" stroke="#8AA3BC" />
      <text id="text_type_g" font-size="10" font-weight="bold" y="0" x="0"> </text>
      <text id="text_tree_g" font-size="10" y="20" x="0"> </text>
      <text id="text_gsize_g" font-size="10" y="40" x="0"> </text>
    </g>
  </g>
  <!--  only to first page auto-zoom -->
  <g onload="def_zoom(270.0,260.0, '156.56.250.226')" > </g>
</svg>
```

## A.2. JSP generování

Následuje fragment JSP skriptu, který je schopen vygenerovat zmíněný SVG obrázek se všemi náležitostmi. Považuji za zbytečné zde komentovat každou funkci či úsek kódu, jelikož vše je zřejmé z vložených komentářů nebo z textu kódu. Jak je patrné z kódu, jedná se pouze o část, která slouží pro hledání uzlů ve smyčce a vykreslování uzlů na požadovanou pozici.

```
<%-- file node.jsp --%>
```

```
<%@page import="javax.servlet.http.HttpServletRequest"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@taglib prefix="myf" uri="../WEB-INF/myTaglib.tld"%>


<c:forEach var="node" items="${node.childs}" varStatus="status" >
 <c:set var="node" value="${node}" scope="request"/>
 <c:set target="${node}" property="x"
  value="${myf:x_position(node.parent.x, node.leftNode.x, status.index, status.first,
 status.last, 30, max_x)}"></c:set>
 <c:set target="${node}" property="y"
  value="${myf:y_position(node.parent.y, 130)}"></c:set>
 <c:if test="${node.x > max_x}"> <c:set var="max_x" value="${node.x}" scope="request" />  </
c:if>
 <c:if test="${node.y > max_y}"> <c:set var="max_y" value="${node.y}" scope="request" />  </
c:if>
<%-- link --%>
 <%-- <c:url value="index.jsp" var="nodeURL">
 <c:param name="ip" value="${node.info.ip}" />
 <c:param name="gip" value="" />
 <c:param name="view_type" value="${URLHandler.view_type}" />
 <c:param name="rate" value="${URLHandler.rate}" />
 <c:param name="mi" value="${URLHandler.mi}" />
 <c:param name="hh" value="${URLHandler.hh}" />
 <c:param name="dd" value="${URLHandler.dd}" />
 <c:param name="mm" value="${URLHandler.mm}" />
 <c:param name="yy" value="${URLHandler.yy}" />
    </c:url>
   <c:url value="index.jsp" var="groupURL">
 <c:param name="ip" value="" />
 <c:param name="gip" value="${node.info.ip}" />
 <c:param name="view_type" value="${URLHandler.view_type}" />
 <c:param name="rate" value="${URLHandler.rate}" />
 <c:param name="mi" value="${URLHandler.mi}" />
 <c:param name="hh" value="${URLHandler.hh}" />
 <c:param name="dd" value="${URLHandler.dd}" />
 <c:param name="mm" value="${URLHandler.mm}" />
 <c:param name="yy" value="${URLHandler.yy}" />
    </c:url>--%>
    <c:set var="nodeURL" value="../index.jsp?ip=${node.info.ip}&amp;view_type=
${URLHandler.view_type}&amp;rate=${URLHandler.rate}&amp;mi=${URLHandler.mi}&amp;hh=
${URLHandler.hh}&amp;dd=${URLHandler.dd}&amp;mm=${URLHandler.mm}&amp;yy=${URLHandler.yy}" />
    <c:set var="groupURL" value="../index.jsp?gip=${node.info.ip}&amp;view_type=
${URLHandler.view_type}&amp;rate=${URLHandler.rate}&amp;mi=${URLHandler.mi}&amp;hh=
${URLHandler.hh}&amp;dd=${URLHandler.dd}&amp;mm=${URLHandler.mm}&amp;yy=${URLHandler.yy}" />
 <%-- end link --%>
 <%-- if it is feedback target in group  --%>
<c:if test="${myf:nodeType(node.class.name) == 'feedback_target'}">

  <a xlink:href="${nodeURL}" target="_top">
  <g id="node${node.info.ip }"
   transform="translate(${node.x},${node.y})">
   <use xlink:href="#feedback_target"
    id="x${node.info.ip }"
    onmousemove="pozice_ft(evt, '${node.info.ip }'), scale_in(evt, '${node.info.ip }')"
    onmouseout="scale_out(evt, '${node.info.ip }'), zmizet_ft()">
   </use>
   <desc id="type${node.info.ip }">Feedback target</desc>
   <desc id="ip${node.info.ip }">${node.info.ip }:${node.port } </desc>
   <desc id="dns${node.info.ip }">${node.info.dns} </desc>
   <desc id="tree${node.info.ip }">${node.feedbackTree} </desc>
   <desc id="gsize${node.info.ip }">${node.nrChilds} </desc>
   <desc id="res${node.info.ip }">${node.resourcesAvailable} </desc>
   <desc id="shared_tree${node.info.ip }">${node.sharedTrees} </desc>
   <desc id="data${node.info.ip }">${node.feedbackDataSize} </desc>
   <desc id="interval${node.info.ip }">${node.feedbackTransInterval} </desc>
   <desc id="activated${node.info.ip }">${node.feedbackActivated} </desc>
  </g>
 </a>
   <line x2="${node.x}" y2="${node.y-23}" x1="${node.parent.x}"
   y1="${node.parent.y+20}" style="stroke:rgb(99,99,99);stroke-width:1" />
```

```
 <jsp:include page="node.jsp"/>
 </c:if>
 <%-- if it is non-group  --%>
 <c:if test="${ (node.parent.nrChilds < group_max_size+1)  }">
  <a xlink:href="${nodeURL}" target="_top">
  <g id="node${node.info.ip }"
   transform="translate(${node.x},${node.y})">

<%-- if is END NODE in non-group --%>
   <c:if test="${myf:nodeType(node.class.name) == 'receiver'}">
   <use xlink:href="#${myf:nodeType(node.class.name)}"
    id="x${node.info.ip }"
    onmousemove="pozice_rec(evt, '${node.info.ip }'), scale_in(evt, '${node.info.ip }')"
    onmouseout="scale_out(evt, '${node.info.ip }'), zmizet_rec()">

   </use>
   <desc id="type${node.info.ip }">End node</desc>
   <desc id="ip${node.info.ip }">${node.info.ip }</desc>
   <desc id="tree${node.info.ip }">${node.feedbackTree} </desc>
   <desc id="data${node.info.ip }">${node.feedbackDataSize} </desc>
   <desc id="interval${node.info.ip }">${node.feedbackTransInterval} </desc>
   <desc id="activated${node.info.ip }">${node.feedbackActivated} </desc>
   <desc id="loss${node.info.ip }">${node.packetLoss}  </desc>
   <desc id="delay${node.info.ip }">${node.delay}  </desc>
   <desc id="jitter${node.info.ip }">${node.jitter}  </desc>
   </c:if>
<%-- --%>
  </g>

 <line x2="${node.x}" y2="${node.y-23}" x1="${node.parent.x}"
   y1="${node.parent.y+20}" style="stroke:rgb(99,99,99);stroke-width:1" />
 </a>
  <jsp:include page="node.jsp"/>
 </c:if>
 <%-- if it is group of receivers --%>

 <c:if test="${(node.parent.nrChilds > group_max_size) }">
 <%-- if is first receiver after nonreceiver or first node in group and receiver  --%>
 <c:if test="${myf:nodeType(node.left.class.name)!='receiver' &&
 myf:nodeType(node.class.name)=='receiver' }">
 <a xlink:href="${groupURL}" target="_top">

  <g id="node${node.parent.info.ip }"
   transform="translate(${node.x},${node.y})">


   <use xlink:href="#receivers_group"
    id="xg${status.index }_${node.parent.info.ip }"
    onmousemove="pozice_g(evt, 'g${status.index }_${node.parent.info.ip }'), scale_in(evt, 'g
${status.index }_${node.parent.info.ip }')"
    onmouseout="scale_out(evt, 'g${status.index }_${node.parent.info.ip }'), zmizet_g()" />

   <desc id="typeg${status.index }_${node.parent.info.ip}">Group of end nodes</desc>
   <desc id="treeg${status.index }_${node.parent.info.ip }">${node.feedbackTree} </desc>
   <desc id="sizeg${status.index }_${node.parent.info.ip }">${node.nrChilds} </desc>
  </g>
  <line x2="${node.x}" y2="${node.y-23}" x1="${node.parent.x}"
   y1="${node.parent.y+20}" style="stroke:rgb(99,99,99);stroke-width:1" />
 </a>
 </c:if>

<c:if test="${myf:nodeType(node.left.class.name)=='receiver' &&
 myf:nodeType(node.class.name)=='receiver' }">
  <c:set target="${node}" property="x" value="${node.left.x}"></c:set>
  </c:if>

  <jsp:include page="node.jsp"/>

 </c:if>
</c:forEach>
```

Pomocí technologie javaBeans jsou ve struktuře JSP volány objekty Javy a metodami objektů nadále pracováno:

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11-flat-20030114.dtd">
<%@page language="java" contentType="image/svg+xml; charset=utf-8"
 pageEncoding="utf-8"%>
<%@page import="javax.servlet.http.HttpServletRequest"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@taglib prefix="myf" uri="../WEB-INF/myTaglib.tld"%>
<%@page import="cz.vutbr.burgetrm.manager.VisualizeManager"%>
<%@page import="cz.vutbr.burgetrm.manager.TreeManager"%>
<%@page import="cz.vutbr.burgetrm.manager.URLHandler"%>
<%@page import="cz.vutbr.burgetrm.model.TreeMember"%>


<jsp:useBean id="treeManager" scope="request"
 class="cz.vutbr.burgetrm.manager.TreeManager" />
<jsp:useBean id="URLHandler" scope="request"
 class="cz.vutbr.burgetrm.manager.URLHandler"
  />
<jsp:useBean id="log" scope="request"
 class="cz.vutbr.burgetrm.model.Log"  />
<jsp:useBean id="hostInfo" scope="request"
 class="cz.vutbr.burgetrm.model.HostInfo"
  />
<jsp:setProperty name="URLHandler" property="*" />

<c:set var="group_max_size" value="3" scope="request" />
<%
 TreeManager Tmng2 = new TreeManager();
 TreeMember node = Tmng2.getRoot(URLHandler);
 request.setAttribute("node", node);
%>
<c:set var="node" value="${node}" scope="request"></c:set>
<c:set var="group_visible" value="n" scope="request" />
<c:set var="nodeParentURL" value="../index.jsp?ip=${node.parent.info.ip}&amp;view_type=
${URLHandler.view_type}&amp;rate=${URLHandler.rate}&amp;mi=${URLHandler.mi}&amp;hh=
${URLHandler.hh}&amp;dd=${URLHandler.dd}&amp;mm=${URLHandler.mm}&amp;yy=${URLHandler.yy}" />
<c:if test="${empty URLHandler.gip}">
<c:set var="group_visible" value="y" scope="request" />
<c:set var="nodeParentURL" value="../index.jsp?gip=${node.parent.info.ip}&amp;view_type=
${URLHandler.view_type}&amp;rate=${URLHandler.rate}&amp;mi=${URLHandler.mi}&amp;hh=
${URLHandler.hh}&amp;dd=${URLHandler.dd}&amp;mm=${URLHandler.mm}&amp;yy=${URLHandler.yy}" />

</c:if>
<c:set var="max_x" scope="request" />
<c:set var="max_y" scope="request" />
```

## A.3. Java

Jak bylo řečeno v předchozím textu této práce, model stromu je reprezentován objektovou strukturou v jazyce Java. Zde je výpis jednotlivých souborů.

## TreeMember.java

```
package cz.vutbr.burgetrm.model;

import java.sql.Timestamp;
import javax.persistence.CascadeType;
import javax.persistence.DiscriminatorColumn;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.Inheritance;
import javax.persistence.InheritanceType;
import javax.persistence.ManyToOne;
import javax.persistence.OneToOne;

@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(length = 1)
abstract public class TreeMember extends EntityBase {
 private TreeMember parent;
```

```
private HostInfo info = null;
private Log log = null;
private TreeMember leftNode;
private TreeMember rightNode;

@ManyToOne(cascade = CascadeType.ALL, fetch = FetchType.EAGER)
public HostInfo getInfo() {
 return info;
}

public void setInfo(HostInfo info) {
 this.info = info;
}

@ManyToOne(cascade = CascadeType.ALL, fetch = FetchType.EAGER)
public Log getLog() {
 return log;
}

public void setLog(Log log) {
 this.log = log;
}

@ManyToOne(cascade = CascadeType.ALL, fetch = FetchType.EAGER)
public TreeMember getParent() {
 return parent;
}

public void setParent(TreeMember parent) {
 this.parent = parent;
}

@OneToOne(cascade = CascadeType.ALL)
public TreeMember getLeftNode() {
 return leftNode;
}

public void setLeftNode(TreeMember leftNode) {
 this.leftNode = leftNode;
}

@OneToOne(cascade = CascadeType.ALL)
public TreeMember getRightNode() {
 return rightNode;
}

public void setRightNode(TreeMember rightNode) {
 this.rightNode = rightNode;
}

private Float x;
private Float y;
private Integer port;
private String nodeType;
private Integer feedbackTree;
private Integer resourcesAvailable;
private Integer sharedTrees;
private Integer feedbackDataSize;
private Integer feedbackTransInterval;
private Integer packetLoss;
private Integer delay;
private Integer jitter;
private Boolean state;
private Timestamp feedbackActivated;

public Integer getPort() {
 return port;
}

public void setPort(Integer port) {
 this.port = port;
}
```

```
public String getNodeType() {
 return nodeType;
}

public void setNodeType(String nodeType) {
 this.nodeType = nodeType;
}

public Integer getFeedbackTree() {
 return feedbackTree;
}

public void setFeedbackTree(Integer feedbackTree) {
 this.feedbackTree = feedbackTree;
}

public Integer getResourcesAvailable() {
 return resourcesAvailable;
}

public void setResourcesAvailable(Integer resourcesAvailable) {
 this.resourcesAvailable = resourcesAvailable;
}

public Integer getSharedTrees() {
 return sharedTrees;
}

public Integer getFeedbackDataSize() {
 return feedbackDataSize;
}

public void setFeedbackDataSize(Integer feedbackDataSize) {
 this.feedbackDataSize = feedbackDataSize;
}

public Integer getFeedbackTransInterval() {
 return feedbackTransInterval;
}

public void setFeedbackTransInterval(Integer feedbackTransInterval) {
 this.feedbackTransInterval = feedbackTransInterval;
}

public Integer getPacketLoss() {
 return packetLoss;
}

public void setPacketLoss(Integer packetLoss) {
 this.packetLoss = packetLoss;
}

public Integer getDelay() {
 return delay;
}

public void setDelay(Integer delay) {
 this.delay = delay;
}

public Timestamp getFeedbackActivated() {
 return feedbackActivated;
}

public void setFeedbackActivated(Timestamp feedbackActivated) {
 this.feedbackActivated = feedbackActivated;
}

public void setSharedTrees(Integer sharedTrees) {
 this.sharedTrees = sharedTrees;
}
```

```
 public Boolean getState() {
  return state;
 }

 public void setState(Boolean state) {
  this.state = state;
 }

 public Integer getJitter() {
  return jitter;
 }

 public void setJitter(Integer jitter) {
  this.jitter = jitter;
 }

 /*
  * public boolean isMainFeedback() { return parent == null; }
  */

 /**
  * @see java.lang.Object#toString()
  */
 /*
  * public String toString() { return new
  * ToStringBuilder(this).append("parent",
  * this.isMainFeedback()).append("parent", this.parent).append("parent",
  * this.parent).toString(); }
  */

 public Float getX() {
  return x;
 }

 public void setX(Float x) {
  this.x = 0.0f;
  this.x = x;
 }

 public Float getY() {
  return y;
 }

 public void setY(Float y) {
  this.y = y;
 }

 public TreeMember() {
 }

}
```

## FT.java

```
package cz.vutbr.burgetrm.model;

import java.io.Serializable;
import java.util.HashSet;
import java.util.Set;
import java.util.Vector;

import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Inheritance;
import javax.persistence.InheritanceType;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
```

```
import javax.persistence.ManyToOne;
import javax.persistence.OneToMany;
import javax.persistence.OneToOne;
import javax.persistence.PrimaryKeyJoinColumn;
import javax.persistence.PrimaryKeyJoinColumns;
import javax.persistence.Table;

import org.apache.commons.lang.builder.ToStringBuilder;
import org.hibernate.*;
import javax.persistence.*;
import org.hibernate.annotations.ForeignKey;
import java.util.List;

@Entity
@DiscriminatorValue("F")
public class FT extends TreeMember {

 private static final long serialVersionUID = -3143430990678751022L;
 //private Integer nrChilds;
 private List<TreeMember> childs = new Vector<TreeMember>();

 public FT() {
  super();
 }

 @OneToMany(mappedBy = "parent")
 public List<TreeMember> getChilds() {
  return childs;
 }

 public void addChild(TreeMember child) {

  child.setParent(this);
  child.setRightNode(null);
  if (childs.isEmpty()) {
   child.setLeftNode(null);
  } else {
   child.setLeftNode(childs.get(childs.size() - 1));
   //this.setNrChilds(childs.size());
  }
  childs.add(child);
 }

 public void setChilds(List<TreeMember> childs) {
  this.childs = childs;
 }


    @Transient
 public Integer getNrChilds() {
  return childs.size();
 }


 /**
  * @see java.lang.Object#toString()
  */
 public String toString() {
  return new ToStringBuilder(this).append("parent", this.getParent())
    .append("info", this.getInfo()).append("childs", this.childs)
    .append("id", this.getId()).toString();
 }

}
```

## Receiver.java

```
package cz.vutbr.burgetrm.model;

import java.io.Serializable;
import java.util.List;
import java.util.Vector;
```

```
import javax.persistence.CascadeType;
import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Inheritance;
import javax.persistence.InheritanceType;
import javax.persistence.OneToMany;
import javax.persistence.PrimaryKeyJoinColumn;
import javax.persistence.PrimaryKeyJoinColumns;
import javax.persistence.Table;

import org.apache.commons.lang.builder.ToStringBuilder;
import org.hibernate.*;
import javax.persistence.*;
import org.hibernate.annotations.ForeignKey;
@Entity
@DiscriminatorValue("R")
public class Receiver extends TreeMember {
 private static final long serialVersionUID = -3143430990678751022L;
 /*not necessary */
 private List<TreeMember> childs = new Vector<TreeMember>();

 public Receiver() {
  super();
 }
 public Vector<TreeMember> getChilds() {
  return null;
 }
 /*not necessary */
 public void setChilds(List<TreeMember> childs) {
  this.childs = null;
 }
 /**
  * @see java.lang.Object#toString()
  */
 public String toString() {
  return new ToStringBuilder(this).append("id", this.getId()).toString();
 }

}
```

## HostInfo.java

```
package cz.vutbr.burgetrm.model;

import java.util.HashSet;
import java.util.List;
import java.util.Set;
import java.util.Vector;

import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToMany;

@Entity
public class HostInfo extends EntityBase{
 //@OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER, mappedBy = "hostInfo")
 private Set<TreeMember> treeMember  = new HashSet<TreeMember>();

 private String ip;
 private String dns;
 public String getIp() {
  return ip;
 }
 public void setIp(String ip) {
```

```
  this.ip = ip;
 }
 public String getDns() {
  return dns;
 }
 public void setDns(String dns) {
  this.dns = dns;
 }
 @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER,mappedBy = "info")
 public Set<TreeMember> getTreeInfo() {
  return treeMember ;
 }
 public void setTreeInfo(Set<TreeMember> treeMember ) {
  this.treeMember  = treeMember ;
 }
}
```

## Log.java

```
package cz.vutbr.burgetrm.model;

import java.util.Date;
import java.sql.Timestamp;
import java.util.HashSet;
import java.util.List;
import java.util.Set;
import java.util.Vector;

import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.JoinColumns;
import javax.persistence.ManyToOne;
import javax.persistence.OneToMany;

@Entity
public class Log extends EntityBase {
 @Id
 @GeneratedValue(strategy = GenerationType.AUTO)
 private Integer id;
 private static final long serialVersionUID = 2554803765420301428L;
 private Set<TreeMember> treeMember = new HashSet<TreeMember>();
 private Date date;

 public Date getDate() {
  return date;
 }

 public void setDate(Date date) {
  this.date = date;
 }

 @OneToMany(mappedBy = "log")
 public Set<TreeMember> getTreeMember() {
  return treeMember;
 }

 public void setTreeMember(Set<TreeMember> treeMember) {
  this.treeMember = treeMember;
 }

}
```

Třídy k udržování paramterů z URL adresy a managementu modelu stromu:

## URLHandler.java

```
package cz.vutbr.burgetrm.manager;
```

```
import java.util.*;
import java.text.SimpleDateFormat;

public class URLHandler {

 private String rate;
 private String default_rate = "30";
 private String gip;
 private String ip;
 private String view_type;
 private String default_view_type = "full";
 private String yy;
 private String mm;
 private String dd;
 private String hh;
 private String mi;
 private Locale locale = Locale.ENGLISH;
 public final String DATE_FORMAT_NOW = "yyyy-MM-dd hh:mm:ss";

 public String getDateInst(String dateFormat) {
  Calendar cal = Calendar.getInstance();
  SimpleDateFormat sdf = new SimpleDateFormat(dateFormat, locale);
  return sdf.format(cal.getTime());

 }

 public boolean isInteger(String nr, Integer max) {
  boolean state = true;
  try {
   int number = Integer.parseInt(nr);
   if (number > max)
    state = false;
  } catch (NumberFormatException ex) {
   state = false;
  }
  return state;
 }

 public String getGip() {
  return gip;
 }

 public void setGip(String gip) {
  this.gip = gip;
 }

 public String getIp() {
  System.out.println("GETIP: " + ip);
  return ip;
 }

 public void setIp(String ip) {
  System.out.println("SETIP: " + ip);
  this.ip = ip;
 }

 public String getView_type() {
  if (null == view_type || view_type.equals(""))
   view_type = "full";
  System.out.println("getViewType: " + view_type);
  return view_type;
 }

 public void setView_type(String aView_type) {
  if (null == aView_type || aView_type.equals("")
    || (!aView_type.equals("full") && !aView_type.equals("simple")))
   aView_type = default_view_type;
  view_type = aView_type;
 }

 public String getRate() {
```

```
 if (null == rate || rate.equals("") || !isInteger(rate, 3000000))
  rate = default_rate;
 return rate;
}

public void setRate(String aRate) {
 if (null == aRate || aRate.equals("") || !isInteger(aRate, 3000000)) {
  aRate = default_rate;
 }
 rate = aRate;
}

public String getYy() {
 if (null == yy || yy.equals("") || !isInteger(yy, 3000)
   || Integer.parseInt(yy) < 2008 || yy.length() < 4) {
  yy = getDateInst("yyyy");
 }

 return yy;
}

public void setYy(String yy) {
 if (null == yy || yy.equals("") || !isInteger(yy, 3000)
   || Integer.parseInt(yy) < 2008 || yy.length() < 4) {
  yy = getDateInst("yyyy");
 }
 this.yy = yy;
}

public String getMm() {
 if (null == mm || mm.equals("") || !isInteger(mm, 12)) {
  mm = getDateInst("MM");
 }
 if (mm.length() < 2)
  mm = "0" + mm;
 return mm;
}

public void setMm(String mm) {
 if (null == mm || mm.equals("") || !isInteger(mm, 12)) {
  mm = getDateInst("MM");
 }
 if (mm.length() < 2)
  mm = "0" + mm;
 this.mm = mm;
}

public String getDd() {
 if (null == dd || dd.equals("") || !isInteger(dd, 31)) {
  dd = getDateInst("dd");
 }
 if (dd.length() < 2)
  dd = "0" + dd;
 return dd;
}

public void setDd(String dd) {
 if (null == dd || dd.equals("") || !isInteger(dd, 31)) {
  dd = getDateInst("dd");
 }
 if (dd.length() < 2)
  dd = "0" + dd;
 this.dd = dd;
}

public String getHh() {
 if (null == hh || hh.equals("") || !isInteger(hh, 23)) {
  hh = getDateInst("HH");
 }
 if (hh.length() < 2)
  hh = "0" + hh;
 return hh;
```

```
 }

 public void setHh(String hh) {
  if (null == hh || hh.equals("") || !isInteger(hh, 23)) {
   hh = getDateInst("HH");
  }
  if (hh.length() < 2)
   hh = "0" + hh;
  this.hh = hh;
 }

 public String getMi() {
  if (null == mi || mi.equals("") || !isInteger(mi, 59)) {
   mi = getDateInst("mm");
  }
  if (mi.length() < 2)
   mi = "0" + mi;
  System.out.println("MI:" + mi + "lenght:" + mi.length());
  return mi;
 }

 public void setMi(String mi) {
  if (null == mi || mi.equals("") || !isInteger(mi, 59)) {
   mi = getDateInst("mm");
  }

  if (mi.length() < 2)
   mi = "0" + mi;
  this.mi = mi;
 }

}
```

## TreeManager.java

```
package cz.vutbr.burgetrm.manager;

import cz.vutbr.burgetrm.filters.HibernateUtil;
import cz.vutbr.burgetrm.model.FT;
import cz.vutbr.burgetrm.model.TreeMember;
import cz.vutbr.burgetrm.manager.URLHandler;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import org.hibernate.*;
import org.hibernate.Query;
import java.util.Date;
import java.util.Locale;
import java.text.DateFormat;

public class TreeManager {

 private TreeMember root = new FT();

 public TreeMember getRoot(URLHandler urlHandler) throws ParseException {
  Date date;
  date = ParseDate(urlHandler.getYy(), urlHandler.getMm(), urlHandler
    .getDd(), urlHandler.getHh(), urlHandler.getMi());
  System.out.println("Hledam info k IP adrese:" + urlHandler.getIp()+" a datu: "+date);
  /* if theres no Ip from URL */
  if ((null == urlHandler.getIp() || urlHandler.getIp().equals("")) && (null ==
 urlHandler.getGip() || urlHandler.getGip().equals(""))) {
   root = getNodeByDate(date);
  }
  else{
   /* if exists IP from url */
   if (null == urlHandler.getGip() || urlHandler.getGip().equals("")) {
    root = getNodeByIpAndDate(date, urlHandler.getIp());
   }
   /* if exists group IP from url */
   else {
    root = getNodeByIpAndDate(date, urlHandler.getGip());
   }
```

```
 }
 return root;
}

/* function for get root node from DB */
public TreeMember getNodeByDate(Date date) {

 Session session = HibernateUtil.getSession();
 Transaction trans = session.beginTransaction();

 System.out.println("Vyparsovane datum  : " + date);
 Query q = session
   .createQuery("from TreeMember tm where tm.log.date <= ? and tm.parent = NULL order by
tm.id DESC LIMIT 1");
 q.setParameter(0, date);

 int ctr = 0;
 System.out.println("POSTUPNE VYPISUJI VSECHNY ZAZNAMY:");
 for (Object o : q.list()) {
  TreeMember tm = (TreeMember) o;
  System.out.println("   ZAZNAM c. " + ctr + ": " + tm);
  ctr++;
 }

 TreeMember p = null;
 if (q.list().size() > 0) {
  p = (TreeMember) q.list().get(0);
 }

 trans.commit();
 session.close();
 // System.out.println("uzel: "+p);

 return p;
}

/* function for get node by IP from DB */
public TreeMember getNodeByIpAndDate(Date date, String ip) {

 Session session = HibernateUtil.getSession();
 Transaction trans = session.beginTransaction();

 System.out.println("Vyparsovane datum  : " + date);
 System.out.println("IP adresa  : " + ip);
 Query q = session
   .createQuery("from TreeMember tm where tm.log.date <= ? and tm.info.ip = ? order by tm.id
DESC LIMIT 1");
 q.setParameter(1, ip);
 q.setParameter(0, date);

 int ctr = 0;
 System.out.println("POSTUPNE VYPISUJI VSECHNY ZAZNAMY:");
 for (Object o : q.list()) {
  TreeMember tm = (TreeMember) o;
  System.out.println("   ZAZNAM c. " + ctr + ": " + tm);
  ctr++;
 }

 TreeMember p = null;
 if (q.list().size() > 0) {
  p = (TreeMember) q.list().get(0);
 }

 trans.commit();
 session.close();
 // System.out.println("uzel: "+p);

 return p;
}

/* to parsing date from URL */
public Date ParseDate(String yy, String mm, String dd, String hh, String mi)
```

```
   throws ParseException {
  Locale locale = Locale.ENGLISH;
  String toParse = yy + "-" + mm + "-" + dd + " " + hh + ":" + mi + ":59";

  DateFormat format = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss", locale);
  Date parsedDate = (Date) format.parse(toParse);

  System.out.println("Datum pred vyparsovanim  : " + yy + "-" + mm + "-"
    + dd + " " + hh + ":" + mi + ":59");
  System.out.println("Datum po vyparsovani1  : " + parsedDate);
  return parsedDate;
 }

}
```

Třída pro výpočet pozice uzlu na plátně vizualizace:

## VizualizeManager.java

```
package cz.vutbr.burgetrm.manager;

import cz.vutbr.burgetrm.model.TreeMember;

public class VisualizeManager extends TreeMember {

 public static float[] xx = new float[100];
 public static int yy = 0;
 public static int i = 0, j = 0;
 // public static float multiplicator = 100;
 public static int mlt_y = 100;

 public static float x_position(String parent_position,
   String left_position, String x, String first, String last,
   String adder, String max_x) {
  float parent_pos = 0, left_pos = 0, add = 0, position = 0, maximum_x = 0;

  if (adder != "") {
   add = Float.valueOf(adder);
  }
  if (max_x != "") {
   maximum_x = Float.valueOf(max_x);
  } else
   maximum_x = 0;

  /*
   * if (parent_position =="" && left_position =="" && first=="true"){
   * max_pos = 0; }
   */
  if (parent_position != "") { // if exist parent set position on
          // paremt plus adder
   parent_pos = Float.valueOf(parent_position);
   // position = parent_pos+add;
   // position = parent_pos+multiplicator;
   // position = max_pos+add;
   // position = maximum_x+add;
   position = maximum_x + add;
  }
  if (first == "true") {
   // reduce distance
  }
  if (left_position != "") {
   left_pos = Float.valueOf(left_position);
   position = left_pos + add;
   if (maximum_x > position)
    position = maximum_x;
   // if (max_pos>position) position = max_pos; //ZDE JE CHYBA
  }
  if (first == "false" && last == "true") {
   // increase distance
  }
  // max_pos = position;
  return position;
```

```
 }

 public static float y_position(String parent_position, String adder) {
  float position = 0, parent_pos = 0, add = 0;

  if (adder != "") {
   add = Float.valueOf(adder);
  }
  if (parent_position != "") {
   parent_pos = Float.valueOf(parent_position);
  }
  position = parent_pos + add;
  return position;

 }

 public static boolean isReceiver(String type) {
  if (type == "cz.vutbr.burgetrm.model.Receiver")
   return true;
  else
   return false;
 }

 public static String nodeType(String type) {
  if (type == "cz.vutbr.burgetrm.model.Receiver")
   return "receiver";
  else
   return "feedback_target";
 }

}
```

## A.4. Javascript

Jak již bylo řečeno, o řízení webové aplikace se stará Javascript, který přistupuje k jednotlivým objektům struktury XHTML a SVG a manipuluje s nimi. Zde jsou uvedeny zajímavější části.

1. Zobrazení informačního textu po najetí myši nad uzel:

```
function pozice_g(evt, id) {
var x = evt.clientX;  //souradnice mysi
var y = evt.clientY;  //souradnice mysi
x=x+10;
y=y+10;
objekt = document.getElementById("extra_text_g");
objekt.setAttribute('transform','translate('+x+','+y+')' );
objekt.setAttribute('visibility','visible' );

var text1 = document.getElementById('type'+id).firstChild.nodeValue;
textuzel = document.getElementById("text_type_g");
var vystup = document.createTextNode(text1);
vystup.replaceData(0,0,"Type: ");
textuzel.replaceChild(vystup,textuzel.firstChild);

var text3 = document.getElementById('tree'+id).firstChild.nodeValue;
textuzel3 = document.getElementById("text_tree_g");
var vystup3 = document.createTextNode(text3);
vystup3.replaceData(0,0,"Feedback tree ID: ");
textuzel3.replaceChild(vystup3,textuzel3.firstChild);

var text4 = document.getElementById('size'+id).firstChild.nodeValue;
textuzel4 = document.getElementById("text_gsize_g");
var vystup4 = document.createTextNode(text4);
vystup4.replaceData(0,0,"Group size: ");
textuzel4.replaceChild(vystup4,textuzel4.firstChild);

evt.stopPropagation(); //stop evt
 return false;
}
```

```
function zmizet_g()
{
objekt = document.getElementById("extra_text_g");
objekt.setAttribute('visibility','hidden' );
 return false;
}
```

2. Skript pro řízení navigace v rámci SVG. Pomocí časovače je zajištěn posun v rámci obrázku.

```
var timer=null;
function translate_view(direction)
{
if (direction !='clear' && timer==null){
timer = window.setInterval("translate('"+direction+"')",50);
var object = document.getElementById('arrow');
object.setAttribute('fill','#b5bbc1' );
object.setAttribute('stroke','#FFFFFF' );
}
else if(direction =='clear') {
window.clearInterval(timer);
timer=null;
var object = document.getElementById('arrow');
object.setAttribute('fill','#FFFFFF' );
object.setAttribute('stroke','url(#arrowXMLID_3_)' );
}
}

function translate(direction, object) {
var x = 0;
var y = 0;
switch(direction){
case "right": {x=-10; y= 0}; break;
case "left": {x=10; y= 0}; break;
case "top": {x=0; y= 10}; break;
case "down": {x=0; y= -10}; break;
}

object = document.getElementById('nodes_group');
//------Thanks of http://www.carto.net/papers/svg/samples/show_coordinates.shtml

//first get transform value of coordinate box
  var curTransform = object.getAttribute("transform");
  curTransform = new String(curTransform); //Wert in ein String umwandeln
  //no fear from Regular expressions, just copy it, I copied it either ...
  var translateRegExp=/translate\(([-+]?\d+)(\s*[\s,]\s*)([-+]?\d+)\)\s*/;
  //This part extracts the translation-part from the transform-string
  if (curTransform.length != 0){
    var result = curTransform.match(translateRegExp);
    if (result == null || result.index == -1){
       var oldTranslateX = 0;
       var oldTranslateY = 0;
    }
    else{
       var oldTranslateX = result[1];
       var oldTranslateY = result[3];
    }
  }
x = parseFloat(oldTranslateX)+parseFloat(x);
y = parseFloat(oldTranslateY)+parseFloat(y);
//-----Thanks
object.setAttribute('transform','translate('+x+','+y+')' );

}
```